NPS-CS-00-006

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Management System for Heterogeneous Networks Final Report Volume I: Project Summary and Papers (Part B)

by

Cynthia E. Irvine H. J. Siegel Viktor Prasanna
Debra Hensgen Timothy Levin

14 April 2000

Prepared for: Defense Advanced Research Project Agency
Information Technology Organization

To Appear in Proceedings of the 2000 Heterogeneous Computing Workshop, May 2000.

Toward Quality of Security Service in a Resource Management System Benefit Function

Cynthia E. Irvine
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943 USA
email: irvine@cs.nps.navy.mil

Timothy E. Levin
Anteon Corporation
2600 Garden Road
Monterey, CA 93940 USA
email: levin@cs.nps.navy.mil

Abstract

Enforcement of a high-level statement of security policy may be difficult to discern when mapped through functional requirements to a myriad of possible security services and mechanisms in a highly complex, networked environment. A method for articulating network security functional requirements, and their fulfillment, is presented. Using this method, security in a quality of service framework is discussed in terms of "variant" security mechanisms and dynamic security policies. For illustration, it is shown how this method can be used to represent Quality of Security Service (QoSS) in a network scheduler benefit function[1].

1 Introduction

Several efforts are underway to develop middleware systems that will logically combine network resources to construct a "virtual" computational system [4] [7] [8] [15] . These geographically distributed, heterogeneous resources are expected to be used to support a heterogeneous mix of applications. Collections of tasks with disparate computation requirements will need to be efficiently scheduled for remote execution. Large parallelized computations found in fields such as astrophysics [14] and meteorology will require allocation of perhaps hundreds of individual processes to underlying systems. Multimedia applications, such as voice and video will impose requirements for low jitter, minimal packet losses, and isochronal data rates. Adaptive applications will need information about their environment so they can adjust to changing conditions.

User acceptance of these virtual systems, for either commercial or military applications, will depend, in part, upon the security, adaptability, and user-responsiveness provided. Several of the projects engaged in building the middleware to create these networks are pursuing the integration of security [6] [10] [23] and quality of service [1] [17] into these systems. The need for virtual networked systems to both adapt to varying security conditions, and offer the user a range of security choices is apparent.

In the network computing context, users or user programs may request the execution of "jobs," which are scheduled by an underlying control program to execute on local or remote computing resources. The execution of the job may access or consume a variety of network resources, such as: local I/O device bandwidth, internetwork bandwidth; local and remote CPU time; local, intermediate (e.g., routing buffers) and remote storage. The resource usages may be temporary or persistent. As there are multiple users accessing the same resources, there are naturally various allotment, contention, and security issues regarding use of those resources.

The body of rules for resolving network security issues is called the network security policy, whereas the body of rules for resolving network contention and allotment comprise a network management policy (which is sometimes taken to include the network security policy). These policies consist of broad policy jurisdictions, such as scheduling, routing, access control, auditing, and authentication. Furthermore, these jurisdictions can be decomposed, typically, into functional requirements, such as, "users from network domain A must not access site B," and "user C must receive a certain quality of service." The network management and security policies, as mapped through the functional requirements, may be manifested in mechanisms throughout the network, including: host computer operating systems, network managers, traffic shapers, schedulers, routers, switches and combinations thereof. As these mechanisms are distributed and are often obscurely related, there has been some interest in the ability to express and quantify the level of support for security policy and Quality of Security Service (QoSS: managing security and security requests as a responsive "service" for which

[1] This work was supported by the DARPA/ITO Quorum program.

quantitative measurement of service "efficiency" is possible) provided in networked systems.

The purpose of this paper is to present the system developed for the MSHN resource management system [8] for describing network security policy functional requirements, to show how QoSS parameters and mechanisms can be represented in such a system, and to provide an example of the use of this system. The remainder of this paper is organized as follows. Section 2 discusses a "security vector" for quantifying functional support of network security policy. Section 3 describes how the security vector can be used for expressing the effects of QoSS in a network-scheduling benefit function; and a conclusion follows in Section 4.

2 Network Security Vector

A *network security policy* can be viewed as an n-dimensional space of functional security requirements. We represent this multidimensional space with a *vector* (S) of security components. Each component (S.c) specifies a boolean functional requirement, whereby the instantiation of a network job either meets (possibly trivially) or does not meet each of the requirements. By convention, a security vector's components are ordered, so they can be referenced ordinally (S.3) or symbolically (S.c). A component may indicate positive requirements (e.g., communications via node n must use encryption) as well as negative constraints (e.g., users from subnet s may not use node n). Components can also be hierarchically grouped. [22] Requirements for a given security service may be represented by one or more components (indicating a service *sub-vector*), and a security service may utilize functions and requirements of other services and their components.

Some jobs can produce output in different *formats*, where a given format (e.g., high resolution video) might be more resource consumptive than another format (e.g., low resolution video). Formats may have differing security requirements, even within the same job. For example, a video-stream format may require less packet authentication [19] , percentage-wise, than a series of fixed images based on the same data. A "quality of service" scheduling mechanism might choose one format for a job over another, depending on varying network conditions (e.g., traffic congestion). Further, adaptive applications may select formats depending upon changing conditions. For example, IPSec, security association (SA) processing using ISAKMP under IKE can permit complex security choices through an SA payload; and the payload recipient may be given transform choices regarding both Authentication Header and Encapsulating Security Protocol [13] .

2.1 Notation

The set of all jobs is represented by *J*. The set of all formats is represented by *I*. The notation S_{ij} identifies a vector containing the portions of *S* that are applicable to job *j* in format *i*, and $S_{ij}.c$ identifies a given component (*c*) of S_{ij}. The relation of *S* to S_{ij} is clarified further, below. The following are some informal examples of security-vector components:

- S.1: user access to resource is equal to read/write; based on table t
- S.2: % of packets authenticated >= 50, <= 90; inc 10
- S.3: clearance (user) = secrecy/integrity (resource)
- S.4: length of confidentiality encryption key >= 64, <= 256; inc 64
- S.5: authentication header transform *in* {HMAC-MD5, HMAC-SHA}
- S.6: packets from domain A to domain B must be encrypted
- S.7: packets from domain A cannot be sent through domain C

Here, "inc 10" indicates that the range from 50 through 90 is quantized into increments of 10, viz: 50, 60, 70, 80, 90. Later, we will need to indicate the number of quantized steps in the component; to do this, one more notational element is introduced, [*S.c*]. In the above examples, [*S.1*] = 1, and [*S.2*] = 5.

2.2 Variant Security Components

When [S.c] > 1, the underlying control program has a range within which it may allow the job to execute with respect to the policy requirement. We refer to this type of policy, and component, as "variant." Security-variant policies may be used within a resource management context, for example, to effect adaptation to varying network conditions. [18] Also, if the policy mechanism is variant, the control program may offer QoSS choices to the users to indicate their preferences with respect to a given job or jobs. Without variant mechanisms, neither security adaptability by the underlying control program nor QoSS are possible, since fixed policy mechanisms do not allow for changes to security within a fixed job/resource environment. While the expression S.c may contain a compound boolean statement (see Section 2.3), by convention it may contain only one variant clause.

2.3 Component Structure

For use in the examples in this discussion, a component has the following composition (see Table 1 for details):

- component ::= boolean expression, variant-range-specifier ; modifying-clause
- boolean _expression ::= boolean_statement [(or | and) boolean_statement]*
- boolean_statement ::= LHS boolean-operator RHS

Note that it is not the focus here to elaborate on a policy representation language. See other efforts and works in progress [2] [3] [5] [16].

A given policy component has a *value* which is a boolean *expression.* This component may also have an *instantiated value* with respect to a specific job and format, which is either "true" or "false." A component has a left hand side (LHS), which is the item that is being tested; of course the LHS has a *value* as well as an *instantiated value.* A component also has a right hand side (RHS), which is what the LHS is tested against, as well as zero or more modifying clauses. Similarly to the LHS, the RHS may have a value (or values) which is dependent on the instantiation of the component.

2.4 Dynamic Security Policies

With a dynamic security policy, the value of a vector's components may depend on the network "mode" (e.g., normal, impacted, emergency, etc.), where M is the set of all modes. There is, conceptually, a separate vector for each operational mode, represented as: S^{mode}. Access to a predefined set of alternate security policies allows their functional requirements and implementation mechanisms to be examined with respect to the overall policy prior to being fielded, rather than depending on *ad hoc* methods, for example, during an emergency.

Initially, every component of S has the same value in each of its modes. Ultimately, components may be assigned different values, depending on the network mode. For example:

- $S^{normal}.a$: % packets authenticated >= 50, <= 90; inc 10
- $S^{impacted}.a$: % of packets authenticated >= 20, <= 50; inc 10
  Note how [*S.a*] changes from 5 to 4 under the impacted mode
- S^{normal}.b: user access to network node = granted; based on table t
- $S^{impacted}$.b: user access to network node = granted; based on table t, OR UID in set of administrators
- $S^{emergency}$.b: UID in set of {administrators, policymakers}

Or, for example, policy makers might decide that the policy should remain in force regardless of network mode:

- $S^{normal}.c = S^{impacted}.c = S^{emergency}.c$: clearance (user) = classification (resource)

Table 1: Simple Component Elements

Element Name	Example S.1	Example S.2
Value	user access to resource r = RW, based on table t	% of packets authenticated >= 50, <= 90; inc 10
Instantiated value	false	true
Value of LHS	user access to resource r	% of packets authenticated
Instantiated value of LHS	W	70
Boolean operator	=	>=
Value of RHS	RW	50
variant range specifier	none applicable	<= 90
Modifying clause	based on table t	inc 10

If a mode is not specified for a component (e.g., "S.a"), normal mode is assumed. This will be the case (i.e., the mode is unspecified) for the remainder of this discussion.

2.5 Refinements to Security Vector

R is the set of resources $\{r_1.. r_n\}$. R_{ij} is the subset of R utilized in executing job j in format i.

T_j is the requested completion time of job j.

Security policies may be expressed with respect to principals (user, group or role, etc.,), applications, data sets (both destination and source), formats, etc., as well as resources in R_{ij}.

The definition of S_{ij} is finally refined as follows: S_{ij} is a vector that is an order-preserving projection of S, such that a component c from S is in S_{ij} if and only if the value of c depends on format i, job j, or any r in R_{ij}. The number of components in a security vector S_{ij} *is* $[S_{ij}]$.

2.6 Summary of Security Vector

S is a general purpose notational system suitable for expressing arbitrarily complex sets of network security mechanisms. S can express variant policies, to allow security expressions of quality of service requests, and can have dynamic security elements to accommodate multiple situation-based policies. In particular, S can represent both (1) static security requirements that may be implemented in a system, as well as (2) the results of running a particular job or set of jobs against such static requirements. The latter usage will be examined in the next section, to express QoSS in a resource management system benefit function.

3 Network-Scheduler Benefit Function

As discussed above, various mechanisms exist for managing contention for, and allotment of distributed network resources. One class of these mechanisms attempts to efficiently schedule the execution of multiple (possibly simultaneous) jobs on multiple distributed computers (e.g., the MSHN project [8] [23] [24] [11] [17]), where each job requires a determinable subset of the resources. Of interest is a benefit function for comparing the effectiveness of such job scheduling mechanisms when they are presented with real or hypothetical "data sets" of jobs.

Jobs are assigned *priorities* for use in resolving resource contention and allocation issues. In some systems, a job's priority may depend upon the particular operating *mode* of the network. [8] Also, the different data formats of a multiple-format job may have different *preferences* (e.g., assigned by a user or "hard wired" as part of the application or job-scheduler database), and different levels of resource usage. [10] [12] A network job scheduler should receive more credit in the benefit function for scheduling high priority and high preference jobs, as opposed to low priority or low preference jobs. That is to say, a scheduler is intuitively doing a better job if important jobs, as judged by priority and preference, receive more attention than unimportant jobs. How much weight the priorities and preferences are given is a matter of network scheduling policy.

For illustration, we introduce a simple benefit function, B, to measure how well a scheduler meets the goals of user preference and system priorities (see [4] , [12] and [21] for other approaches). This function averages preference (p) and priority (P) (use of a priority and preference in measuring network effectiveness have been introduced for the MSHN project [10]).

$$B = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m_j} X_{ij}(P_{ij} + p_{ij})}{2n}$$

Where the characteristic function X is defined for i, j as:

$Xij = 1$ if format i was successfully delivered to job j within time T_j, else 0

and at most one format is completed per job:

$$\forall j \in J\left(\sum_{i=1}^{m_j} X_{ij} \le 1\right)$$

Jobs and formats are defined as above.

P_{j} is the priority of job j

$0 \le P_j \le 1$

The formats for a job are assigned preferences (p) by the user such that:

$0 <= p <= 1$

m_j is the number of {format, preference} pairs assigned for job j

p_{ij} is the preference the user has assigned to format i, job j

the preferences for a job add up to 1:

$$\forall j \in J : \sum_{i=1}^{m_j} p_{ij} = 1$$

This approach assumes that users will assign preference values that correspond to resource usage, since we want the benefit function to indicate a higher value when the scheduler succeeds in scheduling "harder" jobs [12] .

3.1 Adding Security to the Benefit Function

We wish the benefit function to reflect the effectiveness and restrictions of the security policy. First, we define the characteristic security function Z, for i and j:

$Z_{ij} = 1$ if the instantiated value of all components in S_{ij} are true, else 0

The numerator of the benefit function is multiplied by Z, so that no credit is given for jobs that fail to meet the security requirements:

$$B=\frac{\sum_{j=1}^{n}\sum_{i=1}^{m_j} X_{ij}Z_{ij}(P_j+p_{ij})}{2n}$$

Now, for variant components, we wish to be able to give less credit to the scheduler for fulfilling less "difficult" security requirements. One algorithm for expressing this is for each instantiated component (c) in S_{ij} to be assigned a security completion token (g) where $0 \leq g \leq 1$. g_c will indicate the completion token corresponding to component $S.c$. g_c is defined to be the "percentage" of $[S.c]$ met or exceeded by the *instantiated value* of the component's LHS (notated as $S.c''$):

$g_c = S.c'' / [S.c]$

To illustrate the calculation of g_1, for component S.1:

S.1: % of packets authenticated >= 50, <= 90; inc 10

$[S.1] = 5$ (the number of quanta in *S.1)*, $S.1" = 3$ (the job achieves the 3rd quantum (70))

$g_1 = 3/5 = 0.6$

For invariant components, $g = 1$ or $g = 0$. A token (g) whose value is 0 represents a job "failing" the component's security policy. Recall that Z will be 0 when the job/format fails to meet the requirement of any security component, meaning that the function returns no benefit value for that job/format. We introduce a function (A) which averages the tokens of a job:

$A_{ij} = (g_1 + g_2 + .. + g_n)/n$

where n = $[S_{ij}]$ -- the number of components in S_{ij}

and $(0 \leq A_{ij} \leq 1)$

Averages, such as A, over many different elements can tend to minimize the difference that is seen between different data sets. Therefore, we weight the tokens (g) assigned to individual security components to give more credence to components that are "more important" than others, e.g., reflecting network management policies. Each g_n has a corresponding integer weight (w_n), $w_c \geq 0$. So *Aij* becomes:

$A_{ij} = (g_1w_1 + g_2w_2 + .. + g_nw_n)/(w_1 + w_2 + .. + w_n)$

again $(0 \leq A_{ij} \leq 1)$

In the final expression of the network benefit function, A is added to the numerator, providing an average of security, priority and preference.

$$B=\frac{\sum_{j=1}^{n}\sum_{i=1}^{m_j} X_{ij}Z_{ij}(P_j+p_{ij}+A_{ij})}{3n}$$

$0 \leq B \leq 1$, where 1 indicates the maximum scheduling effectiveness.

3.2 Applicability

This technique for quantifying the variant security instantiated by a resource management system is being used in the MSHN project as a factor in representing the effectiveness of its resource assignments [10] . In the MSHN design, the security requirements of network resources (represented by S) are stored in a Resource Requirements Database. This database is consulted during the resource scheduling phase to effectively match jobs to resources. We expect that this measurement technique could also be applied to other resource management systems, such as Condor [15] and Globus [7] .

While different schedulers could be compared with respect to the individual components of B, a summary function such as B would be useful to automate and normalize the comparison process. Additionally, we expect that the security component (viz, A) in an operational system would be complex enough to evade effective manual analysis.

4 Discussion and Conclusion

A security vector has been presented for describing functional requirements of network security policies. It has been shown that this vector can be used for representing security with respect to both quality of service and a network scheduler benefit function.

We are involved in ongoing work to organize the security vector into a "normal form" with sub-vectors or hierarchies corresponding to security policy jurisdictions (such as: access control, auditing, and authentication) and to incorporate a costing methodology for security components, such as can be provided to a resource management system [9] . We are working to develop a means of adjusting the preference expression with a notion of the corresponding resource usage [12] . We are considering how to expand the security benefit function (A) to reflect user quality of security service choices within the range allowed by variant security components, and to reflect performance implications of redundant security mechanisms.

The organizational security policy [20] governing the network may allow individuals or principals representing them to override rules represented by invariant security vector components. For example, a military commander might decide to forgo cryptographic secrecy mechanisms for a job in an emergency (e.g., to improve network performance), even though the system has not been configured with "dynamic" or "variant" security mechanisms, as defined herein. From the perspective of the security vector S and the benefit function, this is a *change* to *or violation of the computer security policy.* It is recommended that this type of policy change be audited.

References

[1] Aurrecoechea, C., Campbell, A., and Hauw, L. "A Survey of Quality of Service Architectures", Multimedia Systems Journal, Special Issue on QoS Architectures, 1996.

[2] Badger, L., Stern, D. F., Sherman, D. L., Walker, K. M., and Haghighat, S. A., "Practical Domain and Type Enforcement for Unix," Proceedings of 1995 IEEE Symposium on Security and Privacy, 1995, Oakland, Ca., pp. 66-77

[3] Blaze, M., Feigenbaum, J., and Lacy, J., "Decentralized Trust Management," in Proceedings of 1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, Ca., pp 164-173

[4] Chatterjee, S., Sabata, B., Sydir, J. "ERDoS QOS Architecture," SRI Technical Report, ITAD-1667-TR-98-075, Menlo Park, CA, May 1998.

[5] Condell, M., Lynn, C. and Zao, J. "Security Policy Specification Language," INTERNET-DRAFT, Network Working Group, July 1, 1999, ftp://ftp.ietf.org/internet-drafts/draft-ietf-ipsec-spsl-01.txt, Expires January, 2000

[6] Foster, I, N. T. Karonis, N. T., Kesselman, C., Tuecke, S. Managing Security in High-Performance Distributed Computing. Cluster Computing 1(1):95-107, 1998.

[7] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[8] Debra Hensgen, Taylor Kidd, David St. John, Matthew C. Schnaidt, H. J. Siegel, Tracy Braun, Jong-Kook Kim, Shoukat Ali, Cynthia Irvine, Tim Levin, Viktor Prasanna, Prashanth Bhat, Richard Freund, and Mike Gherrity, An Overview of the Management System for Heterogeneous Networks (MSHN), 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, Apr.1999

[9] Irvine, C., and Levin, T., Toward a Taxonomy and Costing Method for Security Metrics, Annual Computer Security Applications Conference, Phoenix, AZ, Dec. 1999

[10] Kim, Jong-Kook, Hensgen, D., Kidd, T., Siegel, H.J., St.John, D., Irvine, C., Levin, T., Porter, N.W., Prasanna, V., and Freund, R., A QoS Performance Measure Framework for Distributed Heterogeneous Networks, Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing, Rhodos, Greece, January 2000.

[11] Lee, C. Kesselman, C., Stepanek, j., Lindell, R., Hwang, S., Scott Michel, B., Bannister, J., Foster, I., and Roy, A. The Quality of Service Component for the Globus Metacomputing System. Proc. 1998 International Workshop on Quality of Service, Napa California, pp. 140-142, May, 1998.

[12] Levin, T., and Irvine C., An Approach to Characterizing Resource Usage and User Preferences in Benefit Functions, NPS Technical Report, NPS-CS-99-005

[13] Maughan, D., Schertler, M., Schneider, M., and Turner, J. Internet Security Association and Key Management Protocol, RFC 2408, http://info.internet.isi.edu/in-notes/rfc/files/rfc2408.txt

[14] Ostriker, J., and Norman. M. L., Cosmology of the Early Universe Viewed Through the New Infrastructure. C.A.C.M. 40(11):85-94.

[15] Raman, R., Livny, M., Solomon, M., "Matchmaking: Distributed Resource Management for High Throughput Computing," Proceedings the 7th IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, Ill.

[16] Ryutov, T. and Neuman, C. Access Control Framework for Distributed Applications. INTERNET-DRAFT, CAT Working Group, USC/Information Sciences Institute, draft-ietf-cat-acc-cntrl-frmw-00.txt, August 07, 1998, Expires February 1999, http://gost.isi.edu/info/gaa_api.html

[17] Sabata, B., Chatterjee, S., Davis, M., Sydir, J., Lawrence, T. "Taxonomy for QoS Specifications," Proceedings the Third International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97), February 5-7, 1997, Newport Beach, Ca., pages 100-107

[18] Schantz, R. E. "Quality of Service," to be published in "Encyclopedia of Distributed Computing," 1998.

[19] Schneck, P. A., and Schwan, K., "Dynamic Authentication for High-Performance Networked Applications," Georgia Institute of Technology College of Computing Technical Report, GIT-CC-98-08, 1998.

[20] Stern, D. F., On the Buzzword "Security Policy", Proceedings of 1991 IEEE Symposium on Security and Privacy, 1991, Oakland, Ca., pages 219-230.

[21] Vendatasubramanian, N. and Nahrstedt, K., "An Integrated Metric for Video QoS," ACM International Multimedia Conference, Seattle, Wa., Nov. 1997.

[22] Wang, C. and Wulf, W. A, "A Framework for Security Mea-

surement." Proc. National Information Systems Security Conference, Baltimore, MD, pp. 522-533, Oct. 1997.

[23] Wright, R., Integrity Architecture and Security Services Demonstration for Management System for Heterogeneous Networks, Masters Thesis, Naval Postgraduate School, Monterey, CA, Sept. 1998.

[24] Wright, R., Shifflett, D., and Irvine, C. E., "Security Architecture for a Virtual Heterogeneous Machine." Proc. Computer Security Applications Conference, Scottsdale, AZ, Dec. 1998, pp 167-17

Biographies

Cynthia E. Irvine is Director, Naval Postgraduate School Center for INFOSEC Studies and Research and an Assistant Professor of Computer Science at the Naval Postgraduate School. Dr. Irvine holds a Ph.D. from Case Western Reserve University. She has over thirteen years experience in computer security research and development. Her current research centers on architectural issues associated with applications for high assurance trusted systems, security architectures combining popular commercial and specialized multilevel components, and the design of multilevel secure operating systems.

Timothy E. Levin is a Senior Research Associate at the Naval Postgraduate School Center for INFOSEC Studies and Research. He received a BS degree in Computer and Information Science from the University of California at Santa Cruz, 1981. He has over fifteen years experience in computer security research and development. His current research interests include management and quantification of heterogeneous network security in the context of Resource Management Systems, development of costing frameworks and scheduling algorithms for the dynamic selection of QoS security mechanisms, and the application of formal methods to secure computer systems.

NPS-CS-99-004

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Experiences Using Semi-Formal Methods During Development of Distributed, Research-Oriented, System-Level Software

by

David St. John
Taylor Kidd
Debra Hensgen
Mantak Shing
Shirley Kidd

May 12, 1999

Approved for public release; distribution is unlimited.
Prepared for: Naval Postgraduate School

INITIAL DISTRIBUTION LIST

1. Professor Debra Hensgen 10
   Naval Postgraduate School, Code CS,
   Monterey, CA 93943

2. Dudley Knox Library, Code 52 2
   Naval Postgraduate School
   Monterey, CA 93943-5100

3. Research Office, Code 09 1
   Naval Postgraduate School
   Monterey, CA 93943-5000

REPORT DOCUMENTATION PAGE	Form approved OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 12 May, 1999	**3. REPORT TYPE AND DATES COVERED** TECHNICAL REPORT

4. TITLE AND SUBTITLE EXPERIENCES USING SEMI-FORMAL METHODS DURING DEVELOPMENT OF DISTRIBUTED, RESEARCH-ORIENTED, SYSTEM-LEVEL SOFTWARE	**5: FUNDING** MSHN Project
6. AUTHOR(S) David St. John, Taylor Kidd, Debra Hensgen,Mantak Shing, Shirley Kidd	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School, 833 Dyer Road, Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-99-004
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA / ITO 3701 North Fairfax Drive Arlington, VA 22203-1714	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words.)

The Management System for Heterogeneous Networks (MSHN) is a large, distributed research software system project that began over 18 months ago. The primary goal of MSHN is to develop a framework within which next-generation resource management system (RMS) issues can be investigated. The initial MSHN design was developed using basic object-oriented design principles and the initial prototype was built with object-oriented technology. After building an initial proof-of-concept prototype, we looked to the standardized terms, symbols and diagrams of the Unified Modeling Language to explain the functionality of MSHN to new students and staff members of the development group, as well as interested colleagues outside of the group. As we learned more about the UML and applied it to MSHN in further detail, we found that this semi-formal method not only (I) helped us to communicate MSHN's functionality to others, but also (ii) improved our design, helping us identify bloated packages and opportunities for object re-use, and (iii) enabled us to more easily settle some open questions that had previously been sources of contention among the members of the MSHN team. Additionally, we found the Unified Process to be quite useful as a framework for building research-level, distributed systems software.

14. SUBJECT TERMS software engineering, distributed computing, semi-formal methods, Unified Modeling Language, software development, MSHN, resource management system	**15. NUMBER OF PAGES** 08
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassifed	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT**

NSN 7540-01-280-5800

Standard Form 298 (Rev. 2-89)
Prescribed by

ANSI Std 239-18

Enclosure (6)

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM Robert C. Chaplin
R. Elster
Superintendent
Provost

This report was prepared for Naval Postgraduate School and funded by DARPA.

This report was prepared by:
David St. John, Taylor Kidd, Debra Hensgen,Mantak Shing, Shirley Kidd

David St. John
Staff
Computer Science Department

Taylor Kidd
Associate Professor
Computer Science Department

Debra Hensgen
Associate Professor
Computer Science Department

Mantak Shing
Associate Professor
Computer Science Department

Shirley Kidd
Staff
Computer Science Department

Reviewed by:

Dean D. Boger, Chair
Department of Computer Science

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

Experiences Using Semi-Formal Methods During Development of Distributed, Research-Oriented, System-Level Software

David St. John[§], Taylor Kidd[†], Debra Hensgen[†], Mantak Shing[†], Shirley Kidd[§]

[†]Department of Computer Science
833 Dyer Road, Code/CS
Naval Postgraduate School
Monterey, CA 93943 USA
{kidd, hensgen}@cs.nps.navy.mil

[§]Anteon Corporation
2600 Garden Road
Suite 220A
Monterey, CA 93940 USA
{stjohn, kidds}@cs.nps.navy.mil

Abstract

The Management System for Heterogeneous Networks (MSHN[1]) is a large, distributed research software system project that began over 18 months ago. The primary goal of MSHN is to develop a framework within which next-generation resource management system (RMS) issues can be investigated. The initial MSHN design was developed using basic object-oriented design principles and the initial prototype was built with object-oriented technology (IDL, C++, Java, and CORBA). After building an initial proof-of-concept prototype, we looked to the standardized terms, symbols and diagrams of the Unified Modeling Language (UML) to explain the functionality of MSHN to new students and staff members of the development group, as well as interested colleagues outside of the group. As we learned more about the UML and applied it to MSHN in further detail, we found that this semi-formal method not only (i) helped us to communicate MSHN's functionality to others, but also (ii) improved our design, helping us identify bloated packages and opportunities for object re-use, and (iii) enabled us to more easily settle some open questions that had previously been sources of contention among the members of the MSHN team. Additionally, we found the Unified Process to be quite useful as a framework for building research-level, distributed systems software.

1 Introduction

The Management System for Heterogeneous Networks (MSHN) is a large, distributed, system-level research project. An object-oriented design approach was used in the design and implementation of the first version of MSHN. Because we are developing *research*, system-level software, as opposed to*production*-quality software, we did not initially find it necessary to use semi-formal or formal methods and processes. However, when the MSHN project was selected for integration with other projects, we were specifically requested to describe the high-level functionality and public interfaces of the MSHN components using the Unified Modeling Language (UML) for the benefit of the integration team. After completing this high-level description, and finding much to like about the UML, we began applying it to the lower-level design of the second version of the MSHN components. This use of the UML allowed us to identify several cases of bloated packages and opportunities for

[1] Pronounced "mission"

object re-use that we had not previously identified. We also adopted the Unified Process for MSHN development [4].

Although the Unified Process was very useful, it required some re-configuration. We, as researchers, have different types of requirements from those implementing production software. In the course of describing our experiences with semi-formal methods, we attempt to point out some of these differences and generalize them in a way that we hope will help other system researchers see the value in semi-formal software engineering methods and processes.

In this paper, we describe (1) the status of the MSHN project prior to our application of the Unified Process, and (2) the immediate improvements we were able to make in MSHN following the introduction of a semi-formal method.

2 State of the System Design Before Applying a Semi-Formal Method

The Management System for Heterogeneous Networks (MSHN) is a resource management system for heterogeneous, distributed computing systems that grew out of a previous research project called SmartNet. SmartNet is primarily a scheduling framework designed to assist resource management systems (RMS) in determining the placement of compute-intensive applications in networked systems containing a heterogeneous collection of high-performancecomputers [2].

MSHN's basic goal is to continuously determine the mapping of resources to applications in a dynamic, heterogeneous, distributed environment that provides the best quality of service to a dynamically changing set of applications. The MSHN project is expanding upon SmartNet by addressing the following issues: i) how to handle shared resources such as networks and file servers, (ii) how to transparently monitor resource usage and availability, (iii) how to support adaptive applications, and (iv) how to deliver good quality of service to all applications. MSHN requires a flexible, component-based architecture that supports experimentation. MSHN's initial architecture is shown inFigure 1.

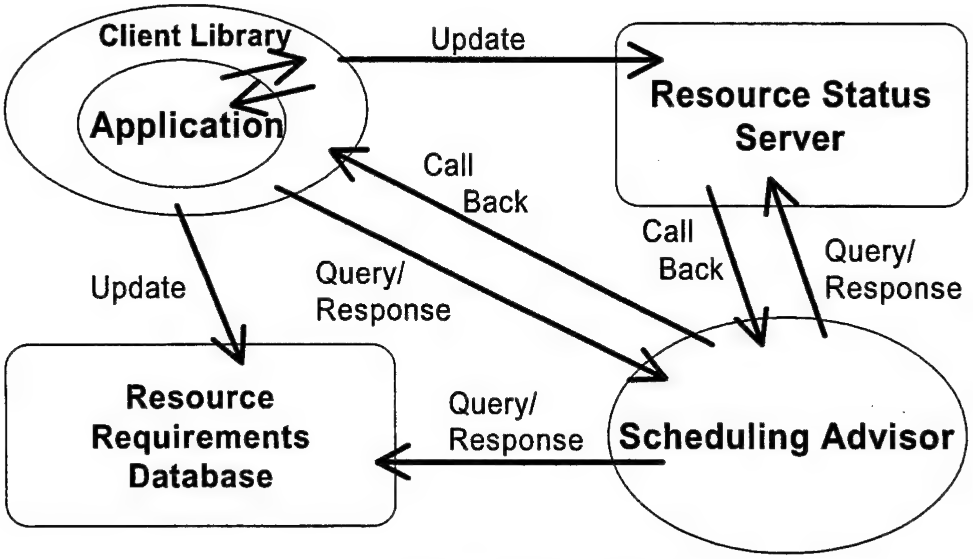


Figure 1. Initial MSHN Architecture.

The original MSHN architecture included four components: the Client Library, the Resource Status Server, the Resource Requirements Database, and the Scheduling Advisor. Each serves a specific function.

The Resource Status Server maintains a quickly changing repository of information pertaining to the current status of the resources that MSHN may assign to processes. The Resource Requirements Database is a database of specific applications' resource requirements, indexed by compute characteristics, a concept pioneered in SmartNet. The Client Library enables an application to transparently interact with the other MSHN components.

The Client Library transparently detects the status of the resources on which an application is running as well as the resource requirements of that application. It reports this information to the Resource Status Server and the Resource Requirements Database, respectively. The Client Library also intercepts requests to start a new application and queries the Scheduling Advisor, which decides where to run the application. It is the Scheduling Advisor's job to determine the best mapping of resources to applications, based upon the current status of the resources, the expected resource requirements of the application, and the requested application's quality of service requirements.

Once the Scheduling Advisor determines where a particular application should be started, the Client Library is responsible for starting the application. Additionally, the Scheduling Advisor sets up callbacks with the Resource Status Server and the Resource RequirementsDatabase so that if any significant changes occur, the Scheduling Advisor will be notified. If the Scheduling Advisor re-computes a new schedule, it contacts the Client Libraries associated with the affected applications. The applications may then adapt to the new schedule in various ways, via their Client Library.

Several teams of people are working on the design and implementation of this project. Two groups, each consisting of a faculty member and several students, tackled the problem of proposing, designing and implementing heuristics that would determine schedules to meet different classes of quality of service requirements. Another group refined the design of the Scheduling Advisor and implemented a prototype of it, working closely with the first two groups. A fourth group examined the question of security as a quality of service attribute. The bulk of the remainder of MSHN's design was to be refined by the authors, along with several staff members and six graduate students. We decided early on that we would describe the interfaces to MSHN's components in IDL. We also decided at an early stage to use CORBA to facilitate interaction among the various components, which, by their very nature, were replicated and distributed. Many students proceeded independently researching individual issues within the MSHN system.

As this research progressed, a new component, the MSHN Daemon, was added, initially with the sole purpose of starting up new applications. Several investigators thought that there was an additional need to determine the status of resources on which no current MSHN tasks were executing, so that functionality was also considered for addition to the MSHN Daemon. Finally, some investigators recognized that the MSHN Daemon might also be responsible for sensing the status of resources that did have MSHN-assigned tasks executing on them.

Additionally, two students worked to develop a generalized application emulator in order to allow us to emulate real applications, on various platforms, without having to install software, purchase licenses, learn the systems, and, in some cases, obtain security clearances.

About halfway through the project, the first MSHN prototype was successfully demonstrated, along with many of the other projects that were also part of the same DARPA program. Based upon this demonstration, and subsequent delivery of documentation, MSHN and several other projects were identified as the basis for components to be combined into a system of systems. At this point, the integration group requested that the MSHN investigators supply a UML description of MSHN[1].

While "retrofitting" a UML description onto the existing MSHN architecture, we decided to also use UML to help us explain MSHN's components to others, including new students, staff members, as well as colleagues working on other systems. Although each of us thought that we understood the interactions between the MSHN components well, before we started documenting them in UML, we found that the UML provided an excellent framework for clarifying points to one another as well as improving our design.

3 Applying a Semi-Formal Method to the System Design

Before we applied any semi-formal methods to MSHN, we had built a prototype of each of the MSHN components. Additionally, to aid in debugging as well as demonstrating MSHN, we had implemented a visualizer that we could use to graphically examine the internals of each of these components. Finally, we had an incomplete design of a MSHN Daemon and some thoughts as to how we were going to increase the functionality of each of our existing prototype components.

Applying a UML description to our existing project not only provided a great framework for communication, but also helped us learn that (i) we were not in total agreement as to eventual functionality, and (ii) our initial design could be improved. This section documents some of our design modifications and identifies some of the improvements that we made.

As would be expected, some of the most striking improvements were made to the components that were added after the initial design, the MSHN Daemon and the MSHN Application Emulator. Due to space constraints, and because there were no substantial modifications made to them, the MSHN Scheduling Advisor, Resource Status Server, and Resource Requirements Database are not described below. The interactions between all MSHN components are, however, illustrated inFigure 2. A good overview of MSHN can be found elsewhere[3].

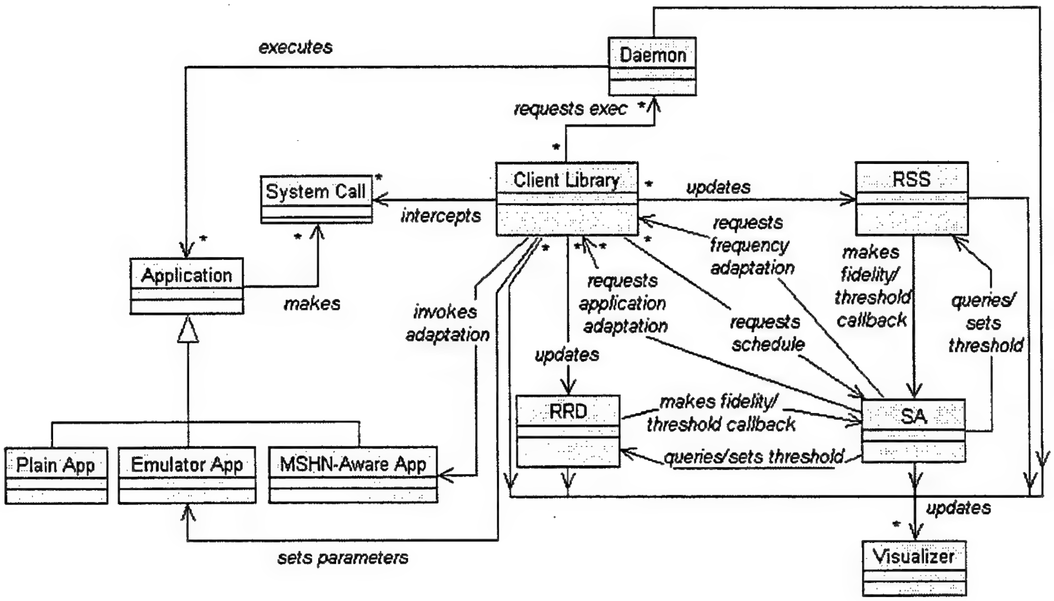


Figure 2. MSHN's Conceptual Model

In the previous section, we described the initial purpose for the MSHN Daemon as well as the functionality that several investigators believed should be embodied in it. Originally, a MSHN Daemon was to reside on each computing resource and was to be used to start up a new application when requested to do so by a Client Library. During design discussions, some members of the design team also wanted the Daemon to aid in determining the status of machines where no applications that were linked with the Client Library were running. These members also believed that the

Daemon may be better equipped to determine the status of some resources on machines that MSHN applications *were* using. The UML description provided a framework that enabled the designers to communicate with one another more clearly. UML descriptions of the MSHN Daemon, Client Library, and Application Emulator components, when simultaneously understood, made it easy to see that keeping the Daemon simple was the proper design decision.

The clarity, which the UML description provided, let us quickly see that the other required functionality, determining the status of resources not currently being used by applications that are linked with the MSHN Client Library, was better obtained by combining the Application Emulator, which had been originally designed for an altogether different purpose, with the Client Library. Interestingly, in these discussions, the Application Emulator became an integral part of the MSHN design, though it also continues to be important in its original role as well. The subsections below document the eventually agreed-upon requirements of the MSHN Daemon, the Client Library, and the Application Emulator and show why the minimally functional MSHN Daemon was the best decision.

Client Library functions. The Client Library provides a transparent interface between each application and the MSHN components. The Client Library transparently intercepts system calls. It collects resource usage and status information, which is forwarded to the Resource Requirements Database and Resource Status Server, respectively. The Client Library also intercepts calls that initiate new processes and consults the Scheduling Advisor for the best place to start each process. It requests (possibly remote) Daemons to execute applications based on the Scheduling Advisor's advice. The Client Library, when notified by the Scheduling Advisor via callbacks, invokes adaptation on MSHN-aware applications. Such adaptation is included under the special case of setting Application Emulator parameters.

Daemon functions. A copy of the MSHN Daemon runs on each compute resource available for use by the Scheduling Advisor. Its sole purpose is to start applications upon request from the Client Library.

Application Emulator functions. The MSHN Application Emulator emulates a running application by stressing particular resources in the same way that real applications do. The Application Emulator serves two purposes. The first (and originally-intended) purpose is as a way of running applications that statistically leave the same resource usage footprint as real applications without the overhead and uncertainty of actually installing, maintaining and running that particular application. The second purpose of the Application Emulator is to perform resource monitoring in the absence of any other MSHN-scheduled applications. Since resource monitoring can be viewed as a kind of application, the Daemon starts one instance of the Application Emulator on each machine, by default, at startup, to monitor resources. The Scheduling Advisor instructs the Application Emulator to monitor more or fewer resources, depending upon the resources that are currently being used and, hence, transparently monitored, by MSHN-scheduled applications.

As alluded to above, in the discussion that accompanies the requirements of the Application Emulator, we note one example of re-use that was discovered during this process, and of which we were completely unaware prior to this analysis. Another interesting point to note is the complexity of the Client Library. By comparing its complexity to that of other components, we were able to determine that, in order to complete that component, we would need to devote substantially more man-power than we had previously devoted to that component.

4 Conclusions

The Management System for Heterogeneous Networks (MSHN) is a large, ambitious, system-level research project. MSHN's major goal is to determine the best way to design and implement many features of a resource management system (RMS) that is able to deliver good quality of service to a mixture of applications with different requirements. Because MSHN is such a large, complex project, it requires the expertise and experience of many different investigators from distributed operating systems, resource management systems, computer architecture, theory of algorithms, control theory, stochastic processing, and dynamic programming.

Although MSHN's investigators had learned quite a lot from designing and implementing previous research resource management systems, their initial design of MSHN did not identify all of the components whose functionality deserved to be isolated and further investigated. One of those components, the MSHN Daemon, was identified because it was determined that implementation of a particular functionality, starting applications on new machines, would otherwise be too machine- and operating system-dependent. Another component, the Application Emulator, which later turned out to beintegral to the overall functionality of MSHN, was designed initially for simply testing the system.

The Unified Modeling Language (UML) and a modified version of the Unified Process were applied halfway through the MSHN project, but still proved to be quite valuable. Several cases of object re-use and bloated packages were discovered. Perhaps, more importantly, UML and the Unified Process provided a framework that allowed the experts from the various fields to communicate more easily their experience to one another. We believe strongly that the MSHN project benefited from these semi-formal methods, and we intend to use them again in other projects, earlier and more often.

5 References

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, Reading, MA, 1999.

[2] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Kieth, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *Proc. 7th IEEE Heterogeneous Computing Workshop*, March 1998, pp. 184-199.

[3] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: the Management System for Heterogeneous Networks," *8th Heterogeneous Computing Workshop* (*HCW '99*), April 1999.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, Reading, MA, 1999.

A QoS Performance Measure Framework for Distributed Heterogeneous Networks

Jong-Kook Kim[1], *Debra A. Hensgen*[2],
Taylor Kidd[2], *Howard Jay Siegel*[1], *David St. John*[3], *Cynthia Irvine*[2],
Tim Levin[3], *N. Wayne Porter*[2], *Viktor K. Prasanna*[4], and *Richard F. Freund*[5]

[1]Purdue University
ECE School
West Lafayette, IN 47907-1285, USA
{kim42, hj}@purdue.edu

[2]Naval Postgraduate School
CS and ECE Departments
Monterey, CA 93943, USA
{irvine, nwporter}@cs.nps.navy.mil
{hensgen, kidd}@acm.org

[3]Anteon Corporation
2600 Garden Rd., Ste. 220A
Monterey, CA 93940, USA
{stjohn, levin}@cs.nps.navy.mil

[4]University of Southern California
Department of EE-Systems
Los Angeles, CA 90089, USA
prasanna@ganges.usc.edu

[5]NOEMIX Inc.
1425 Russ Blvd., Ste. T-110
San Diego, CA 92101 USA
rffreund@noemix.com

Abstract

In a distributed heterogeneous computing environment, users' tasks are allocated resources to simultaneously satisfy, to varying degrees, the tasks' different, and possibly conflicting, quality of service (QoS) requirements. When the total demand placed on system resources by the tasks, for a given interval of time, exceeds the resources available, some tasks will receive degraded service or no service at all. One part of a measure to quantify the success of a resource management system (RMS) in such a distributed environment is the collective value of the tasks completed during an interval of time, as perceived by the user, application, or policy maker. The Flexible Integrated System Capability (FISC) ratio introduced here is a measure for quantifying this collective value. The FISC ratio is a multi-dimensional measure, and may include priorities, versions of a task or data, deadlines, situational mode, security, application- and domain-specific QoS, and dependencies. In addition to being used for evaluating and comparing RMSs, the FISC ratio can be incorporated as part of the objective function in a system's scheduling heuristics.

1. Introduction

In many distributed environments, the tasks that are executing have different quality of service (QoS) requirements. These different QoS requirements impose different machine and resource requirements. Furthermore, these tasks may require input data from a variety of distributed sources. Mixed-machine heterogeneous computing (HC) environments provide a distributed suite of different types of machines, connected by diverse types of networks, to meet the varied computational and input requirements of such task mixtures (e.g., [4, 8, 18]). The goals of a resource management system (RMS) in an HC environment are to assign communication, computation, and other resources in an attempt to satisfy users' requests, which may require different types and levels of QoS. Due to limitations on resource availability in certain situations, some tasks will receive degraded service or none at all. A performance measure is needed to quantify the collective value of the tasks completed during an interval of time, as perceived by the user, application, or policy maker. This measure can be a part of a metric to assess the success of an RMS (other parts may include execution time, ability to work with different operating systems, and user interface). This research describes attributes that can be combined to derive such a performance measure, and provides a way to combine them.

This research is supported by the DARPA/ITO Quorum Program, by the DARPA/ISO BADD Program and the Office of Naval Research under ONR grant number N00014-97-1-0804, and by the DARPA/ITO AICE Program under contract number DABT63-99-C-0010.

The Flexible Integrated System Capability (FISC) ratio is introduced in this research. It is a multi-dimensional performance measure, and may include priorities, versions of a task or data, deadlines, situational mode, security, application- and domain-specific QoS, and task dependencies. This FISC performance measure combines these attributes to determine the collective value of the tasks completed during an interval of time on a distributed computing system.

The FISC ratio can be used for the postmortem analysis of a system to determine the best scheduling heuristic for a given environment. It can also compare, via experimentation or simulation, the effectiveness of changing the resources available in a given distributed system. Furthermore, the FISC ratio can be incorporated as part of the objective function in a system's scheduling heuristics.

This research is part of three related DARPA activities: the DARPA Quorum-sponsored Management System for Heterogeneous Networks (MSHN) project [10], the DARPA Battlefield Awareness and Data Dissemination (BADD) program [22, 7], and the Agile Information Control Environment (AICE) program [1]. The goal of the Quorum MSHN project is to design, prototype, and refine a distributed RMS that leverages the heterogeneity of resources and tasks to deliver the requested QoS. In the Quorum environment, it is sometimes the case that not all tasks requested can achieve their most preferred QoS. Thus, there must be a performance measure that can determine a collective value of the set of tasks that were completed in a given time interval by a particular resource management strategy.

One aspect of the BADD and AICE programs involves designing a scheduling system for forwarding (staging) data items prior to their use as inputs to a local application in a wide area network (WAN) distributed computing environment. The BADD and AICE systems are similar to the Quorum environment in that, in some situations, not all data requests will be satisfied with their most preferred QoS by their deadline. Thus, the goal of the scheduler is to satisfy a set of requests in a way that has the greatest collective perceived value.

The performance measure described in this research is used to evaluate, for a given interval of time, the total value of tasks completed in the MSHN project and the total value of data received in the BADD and AICE programs. In this sense, the set of completed tasks for the MSHN project is equivalent to the set of satisfied data item requests for the BADD and AICE programs. A major difference between MSHN and BADD/AICE is that in MSHN tasks are assigned to resources by the RMS. In the BADD/AICE project, task assignments are given and fixed *a priori*, but the movement of data to the tasks must be scheduled. Throughout the rest of this paper, task will be used to represent a user's process execution in the Quorum MSHN context, and a user's request for a data item in the BADD/AICE context. While this research is motivated by military applications, the FISC ratio can be adapted for other environments, such as industrial intra-nets, government agency intra-nets (e.g., NASA), and computational grids [9].

The next section provides a brief overview of some of the literature related to this work. In Section 3, several examples of individual QoS requirements are presented. These requirements may be considered when formulating the performance measure to be used in building and assessing RMSs. Section 4 shows how all the example QoS attributes can be combined into a single measure. This section also presents a baseline for that measure and discusses a generalized form of the performance measure. An example of how this measure would be instantiated in a military C4I (command, control, communications, computers and intelligence) environment is provided in Section 5. The last section presents a brief summary of this research.

2. Related Work

The FISC performance measure discussed here embodies parameters that are considered important in scheduling tasks and communications on a distributed computing system. There is much literature on scheduling and mapping of tasks, messages, and data items; in this section, some of the literature is described.

An optimistic priority-based concurrency control protocol that schedules active transactions with firm-deadline in real-time database systems is described in [12]. This protocol combines forward and backward validation processes to control more effectively concurrent transactions with different priorities. The protocol is designed such that deadlines of higher priority transactions have a greater probability of being met than those of lower priority transactions. While this is also the case for MSHN and BADD/AICE, the research presented here includes other attributes that are important in evaluating the overall value of the tasks completed.

In [17], priority is used for the mapping, adjustment, and dropping of messages. The priority of a task is adjusted by the length of time it was blocked by a higher priority message and tardy messages that already missed their deadlines are dropped. This Least-Laxity-First priority mapping gives an improved missed deadline ratio, which is the rate of messages missing their deadlines. In [17], priority and deadline is used as measures of performance but the research effort does not consider other QoS attributes used in heterogeneous distributed networks.

An algorithm that allows transmission of messages belonging to several classes of situational mode is presented in [20]. The algorithm takes into account the actual priority of a message in a given class. This algorithm rejects packets with deadlines shorter than a minimum acceptance deadline defined for a particular class. There can be more than one simple deadline. This and other important QoS attributes are discussed in detail.

Data staging, an important data management problem for a distributed heterogeneous networking environment, is discussed in [28]. This was done under the assumption that each requested data item is associated with a specific deadline and priority. The research presented here generalizes part of the objective function used in [28] to include more types of deadlines and other QoS attributes.

From the works mentioned, parameters such as task priority and deadline appear to be important attributes for making scheduling decisions regarding tasks, messages, or data items. A measure of the overall value is needed that can be used in an objective function to compare and analyze the algorithms, protocols, and heuristics while incorporating all the QoS parameters used. The works mentioned above consider only a few of the QoS parameters that are being used in a distributed system. Other parameters, e.g., accuracy, precision, and security, that are QoS requirements and part of the users' requests, must be included in the performance analysis. These and other QoS parameters that affect the overall value of requests satisfied are discussed in this research.

The research on a performance measure presented here builds on and extends a body of earlier work in this field. Some examples are mentioned here.

The ERDoS project [5] describes an objective function for optimizing the effectiveness of its QoS scheduling mechanisms in meeting clients' needs. This function reflects the benefit received by the user and a weight assigned to each user application. An approach where requested QoS is taken into account when scheduling computational resources in a network is presented in [19]. The model proposed a benefit function that uses application deadlines and application priorities as metrics in maximizing the total benefit for the applications. Multiple versions of a task in addition to priorities and deadlines in the objective function are described in [14]. The FISC performance measure presented in this paper serves a purpose similar to the ones in [5, 19, 14]. However, the research presented here provides a more detailed description of a measure using more parameters, so that it can be used to find the performance of an effective schedule in the Quorum MSHN and BADD/AICE environments. Furthermore, the QoS input specification for ERDoS [24] accounts for only two specific security parameters (confidentiality and integrity), whereas the security component of the performance measure in this research can describe an arbitrarily complex set of security features.

The FISC ratio is similar to the utility function described in [31] in that both measure the system value. The utility function in [31] calculates the system value by summing the value of resources allocated and resources reserved depending on the duration of a job, the deadline of a job, and the price of allocated time slots. The FISC ratio uses a conceptually similar calculation, but formulates it differently and includes priorities and other QoS measures to calculate the collective value of task completed.

A security policy that allows a percentage of packets authenticated to vary with network load is described in [25]. This type of policy can be accommodated with the variant components included in the FISC security vector (see Subsection 3.5). While the FISC security vector contains a set of Boolean security policy statements, it does not specify a general-purpose language for these statements. Related work on network security policy specification languages can be found in [2, 3], and works in progress [6, 23]. A framework for quantifying the strength of a set of security mechanisms is described in [30], where high-level static security properties can be decomposed hierarchically. However, in [30] the approach cannot accommodate the measurement of how well an executed task meets the security requirements of its environment. Nor does [30] account for variant security policies or mechanisms.

The FISC ratio measures the effectiveness of a schedule in terms of tasks completed and percentage of their requirements satisfied. In [26], other types of attributes for evaluating different QoS and RMS services in distributed real-time systems are investigated. Some examples include survivability (fault-tolerance), openness (open architecture), and testability (easily testable and verifiable). These are outside the scope of the FISC performance measure, which focuses on the collective worth of the completed tasks.

3. Example QoS Attributes

3.1. Overview

Examples of attributes that need to be considered in the FISC performance measure will be described in this section. The attributes discussed include user-based and application-based priority levels, user preferences for different versions of a task, deadlines, security, other application- and domain-specific QoS attributes, and task dependencies. The attributes used in any given situation must have interpretations that are meaningful to the users, policy makers, and the RMS. A method of determining the weightings, worths, functions, and factors are described in this section.

3.2. Priorities

Policy makers determine the number of priority levels and assign some semantic meaning to each priority level, such that each level qualitatively reflects the relative importance (e.g., high, medium, and low). The policy makers may be the commanders in a military environment or executives in a corporation. Policy makers may assign different users, or classes of users, restricted ranges of priority levels that can be assigned to their tasks. Alternatively, a task itself could have an immutable priority level assigned to it by the policy makers. Each priority level will then be given a weight that can be calculated by a priority weight function, which is predetermined by policy makers, described later in this section.

Priority levels with relative, quantitative weightings should be incorporated in scheduling systems so that a task with a higher importance will have a higher probability of meeting its QoS requirements. Application users and system builders often assign an arbitrary numbering scheme to priority levels that does not meaningfully quantify the relative importance of one priority level to another. Such a scheme, therefore, cannot be used alone in the measure. A more meaningful weight must instead be assigned to each priority level so that the relative importance can be reflected in the performance measure.

The relative importance (weighting) of priority levels may vary depending upon the situational mode. For example, there may be military modes of peace and war. In the peace mode, it might be just as important to complete ten low priority level tasks as to complete one high priority level task. However, in the war mode, one high priority level task might be more important than 1000 medium priority level tasks. This dependency can be indicated in the performance measure by expressing the weight of all priority levels as a function of the situational mode.

It is assumed that the FISC ratio is being used to compute the value of a subset of tasks successfully completed, during some time interval, from a set of t tasks that have been requested. Let the priority level (e.g., high, medium, low) of task j ($0 \leq j < t$) be p_j, and let m be the situational mode. The priority weight function $\pi(p_j, m)$ calculates the weight of p_j given m. The weight assigned to a priority level may be considered to be the maximum value of completing the corresponding task, if all of the task's specified QoS requirements are completely satisfied.

3.3. Versions

A task may exist in different versions, each with its own resource requirements. Because available resources will vary dynamically, it may not be possible to complete the most desired version of a task. For example, a user requesting a map application may most desire a 24-bit color, three-dimensional topographical map. However, if this cannot be given to the user due to limited resources, the user would rather have a black and white, two-dimensional map than nothing at all. When a user's first choice of a task version cannot be completed, a method for choosing an alternative version is needed. Having multiple versions of a task is related to the precision and accuracy parameters discussed in [24], in the sense that each version of a task may have different accuracy and precision.

For each version of a given task, in a given situational mode, a worth (preference) relative to the other versions will be indicated by the application developer, the user, and/or the policy makers. In the above example, the black and white version may only be worth 75% of the color version to the user. When selecting a version of a task to execute, an RMS's scheduling algorithms must consider this worth and the task's resource requirements as well as the availability of these resources. For example, one version may not be viable because its bandwidth requirement is too high.

	worth			normalized worth		
Task\version	0	1	2	0	1	2
0	1	1	8	.125	.125	1
1	25	35	40	.625	.875	1
2	.2	.3	.5	.4	.6	1
3	.1	.2	.7	.143	.286	1

Table 1: Worths and normalized worths that indicate preference for each version of a task.

The worths assigned to different versions of a task must be normalized so that there is a consistent scheme for evaluating worths across tasks and versions. For example, assume that all factors except version worths are equal across a set of tasks. The user can specify any number for the worth of a version as shown in Table 1. Therefore without a normalization procedure, a version with the smallest worth of a certain task can always be chosen for processing ahead of other tasks for no logical reason. For example, the version 0 of task 1 that is the version with the lowest worth of task 1 would be chosen over the version 2 of task 0 that is the version with the highest worth for task 0. In extreme cases, worths that are not normalized could make the priority irrelevant depending on how priorities and worths of version interact.

To avoid this type of anomalous behavior, worths are normalized as follows. Assume there are I_j number of versions for each task j. Let v_{ij} be the i-th ($0 \leq i < I_j$) version of task j. Let $w_{ij}(m)$ be the worth the user assigns to i-th version of task j given m, the situational mode. Example $w_{ij}(m)$ values are provided in Table 1. One approach to the normalization problem is to divide each indicated worth of a task version by the largest worth for that task, resulting in the normalized worth as shown in Table 1. The normalized worth (η_{ij}) of $w_{ij}(m)$ is then given by

$$\eta_{ij} = w_{ij}(m) \,/ \max_{0 \leq a < I_j} w_{aj}(m). \qquad (1)$$

As shown in Table 1, all worths for all versions of each task are normalized by the version with the largest worth. Therefore, the version with the largest worth of each task will have a normalized worth of 1 and the rest of the versions will have normalized worths that are relative to the version with the largest worth. This is the reason why the indicated worths for each task need not sum to one.

Another approach to the normalization would be to divide each version's worth by the total sum of the version worths of the task. This would not guarantee equal value for the most preferred version of each task. Furthermore, this approach would allow a greedy person to obtain a higher value for his/her preference for the version with the largest worth. For example, consider task 0 and task 1 in Table 1. If this alternative approach is used, the normalized worth for task 0 would be .1, .1, and .8, while for task 1 it would be .25, .35, and .4. This means that, even if task 0 and task 1 have the same priority, the largest worth version of task 0 is worth more than the largest worth version of task 1, which should not be the case. In extreme cases, priorities may be irrelevant depending on how priorities and worth of versions interact.

3.4. Deadlines

Many tasks in typical heterogeneous computing environments have deadlines associated with them. Frequently, due to limited resources and the multiplicity of tasks sharing these resources, not every task can be completed by its deadline. Three types of deadlines will be considered for the i-th version of task j: earliest (e_{ij}^d), soft (s_{ij}^d), and firm (f_{ij}^d). These three deadlines are illustrated by example in Figure 1. The deadline attribute discussed here is related to the timeliness parameter in [24].

The earliest deadline (e_{ij}^d) is the time when the system is ready to complete a task, based upon, for example, the availability of the required input data. Therefore, if a task completes before the earliest deadline the task has 0 value.

The soft deadline (s_{ij}^d) is the time by which a task must complete to be of full value to the user [27]. If a task is completed between the earliest deadline and the soft deadline, then the task will have its maximum value.

A task that is completed after its firm deadline (f_{ij}^d) will have 0 value, because the task will be of no use after that deadline [15, 27]. For example, if a task that shows a map of an area completes after a mission is finished, then it will have no value. If a task completes between its soft and firm deadline, then it will have some fraction of its maximum possible value. For each task, the fraction of total value for each point between the soft and firm deadlines, and the time between the soft and the firm deadlines, may be a function of the situational mode. For example, during war mode, the soft and firm deadlines may be identical.

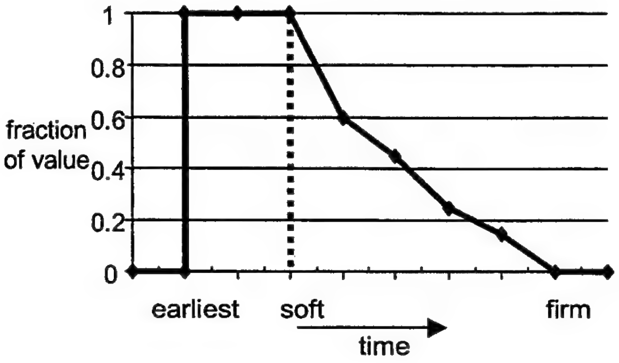


Figure 1: The deadline coefficient graph (non-increasing) shows the variation in the value of a task with various deadlines.

Let τ_{ij} be the time that the i-th version of task j actually completes. The deadline function $\delta_{ij}(\tau_{ij}, m)$ assigns a fraction of the maximum value of the i-th version of task j based on m, τ_{ij}, e_{ij}^d, s_{ij}^d, and f_{ij}^d, where $0 \leq \delta_{ij}(\tau_{ij}, m) \leq 1$. If no version of a task is completed, $\delta_{ij}(\tau_{ij}, m) = 0$ for all versions of the task. The deadlines may be the same for all versions of a certain task.

3.5. Security

User and task security requirements are met by "security services." Overall network security can be viewed as a multi-dimensional space of security services. This multi-dimensional space can be represented with a vector ($\underline{S}$) of security components, where the functional requirement for each component is specified by a Boolean statement for each given situational mode. Both resources and tasks may have multiple security components [11].

The instantiation of a network task either meets, or does not meet, each component's requirement. For example, consider the i-th version of task j. Let $\underline{R}_{ij}$ be an ordered set of resources utilized by v_{ij} and let $\underline{S}_{ij}$ be a sub-vector of vector S. A component $\underline{c}$ in S is in S_{ij} if and only if c depends on v_{ij} or on an element of R_{ij}, and is denoted $\underline{S_{ij}.c}$. Let $\underline{Z}_{ij}$ be 1 if the instantiated Boolean value of all c in S_{ij} is true and 0 otherwise. Z_{ij} corresponds to the required security attributes. This means that if minimum security requirements are not met, then no benefit is accrued from executing v_{ij}.

Additionally, some security components of a task can be variant in that they allow a range of behavior with respect to a requirement (e.g., length of cryptography key may vary between 40 and 256). For variant components, the user may request a particular value or permit a choice from a component's defined range. The RMS must select a specific value within the user's range while considering resource availability, for the completed task to have a non-zero value. The measure will give only partial credit for a task completed with less than the most secure value in the defined range. Thus, additional benefit is accrued if increased security is selected within the range.

The desire to provide *adaptable* security motivates the inclusion of variant security components in the system [16]. Thus, security affects the performance measure when components are variant. For example, assume the percentage of authenticated packets can range between 50% and 90% in increments of 10%. The increment quantizes the range. Let $[\underline{S_{ij}.c}]$ be the number of quanta in $S_{ij}.c$ (in the above case this is five) and $g_{ij}.\underline{c}$ be the fraction of c in S_{ij} satisfied. If a task achieves the third quantum (70%), then $g_{ij}.c$ is $3/[S_{ij}.c] = 3/5 = 0.6$. Suppose $\underline{n}$ is the number of security components in S_{ij}. To quantify the effectiveness of the RMS in providing variant security, let $\underline{A}_{ij}$ be the sum of all $g_{ij}.c$ divided by n as shown in Equation 2.

$$A_{ij} = (\sum_{c \in S_{ij}} g_{ij}.c)/n \qquad (2)$$

The above is just one possible way to combine the values of these security components. For example, the $g_{ij}.c$ values in A_{ij} equation can have relative weightings for given m. Thus, if the military situation changes from peace to war, encryption rate would be considered relatively more important and might be given a high relative weighting.

The overall security factor is defined as: $\underline{\sigma}_{ij} = A_{ij} \times Z_{ij}$, where $0 \leq \sigma_{ij} \leq 1$. It indicates how the value of a task may be degraded due to lack of its most desirable security services.

3.6. Other Application- & Domain-Specific QoS

There is a multi-dimensional space of application- and domain-specific QoS attributes (e.g., jitter level, frame rate, bit error rate) that can also be represented by a vector. There will be zero or more such components per task or data set. These components can be specified in the same way that security was represented. The FISC ratio will use $\underline{\alpha}_{ij}$ to denote such QoS components, analogous to the security factor σ_{ij} [13].

3.7. Dependencies

There are many possible types of inter-task dependencies, e.g., for the MSHN environment, consider a task whose only function is to generate data for other tasks (descendants). There may be an inter-related set of such tasks. If there is a descendant along a path of tasks that does more than just generate data for a subsequent task and if this descendant completes its execution, then the tasks that did nothing more than generate data for this particular task will have value, otherwise they will not. This is because the end user that submitted a task for completion will acknowledge the task to be finished only when the actual results can be determined. If there is no descendant task that produces a result important to a user, all predecessors are worthless to the user.

The first task in a path that does more than generate data for a subsequent task will be known as a required associate of all of its predecessors. This is one way of determining the value of ascendants that only generate data for subsequent tasks. The variable $\underline{\rho}_{ij}$ will be used to represent whether a required associate of a given task completes. That is, if at least one required associate of a given task completes, then $\rho_{ij} = 1$, otherwise $\rho_{ij} = 0$. For the BADD/AICE environment, a similar definition of required associates can be established based on the set of data requested by a given application.

4. Performance Measure

4.1. FISC Ratio

The question of what it means to "provide good QoS" to a mixture of applications in a distributed system is considered in this section. In general, it is a difficult problem to determine whether a distributed system has delivered good service to a mixture of applications with different priorities, receiving different degraded level of QoS etc. A meaningful way to combine the QoS attributes previously discussed is motivated and proposed in this section.

First, consider a set of tasks, possibly with different priorities and different deadlines, where the firm and soft deadlines are identical. Assume also that each task only has a single version. Therefore, the initiators of a task need not specify a preference for a version. It is obvious that a system with sufficient resources will complete all tasks by their deadlines. However, if resources are insufficient and tasks cannot complete by their deadlines, then the RMS will complete the tasks with the higher priorities by their deadline. Such an RMS maximizes

$$\sum_{j=0}^{t-1} \pi(p_j, m) \times \delta_j(\tau_j, m), \tag{3}$$

where $\delta_j(\tau_j, m)$ is 1 if the task is completed by its deadline and 0 otherwise. Even when the deadline is more general, as described in Subsection 3.4, it is easy to see that the same criterion should be maximized.

Now consider the same situation, except with tasks having multiple versions. Again, if there are sufficient resources to complete the most preferred versions of each task, then an RMS will allocate resources and initiate those most preferred versions. However, if resources are insufficient for completing the most preferred versions of all tasks, the RMS must consider using less preferred versions for some tasks. Typically, this will result in a reduction of both the value of completing the task and the amount of the resource consumed. Thus, for the resources available the RMS should maximize

$$\sum_{j=0}^{t-1} \pi(p_j, m) \times [\max_{0 \le i < I_i} \delta_{ij}(\tau_{ij}, m) \times \eta_{ij})]. \tag{4}$$

If only one version of a task is completed, δ_{ij} for all other versions would be 0. Therefore, the max function would be used to indicate whether a version of a task is completed within its deadline.

Similarly, the RMS must consider any reductions in value of a task's completion that is a result of not receiving all required and desired security, and other application- and domain-specific QoS requirements. Furthermore, any dependencies must be obeyed for a task's completion to have any value. One way to accomplish this is to multiply Equation 4 by ρ_{ij}, σ_{ij}, and α_{ij}:

$$\sum_{j=0}^{t-1} \pi(p_j, m) \times [\max_{0 \le i < I_i} [\eta_{ij} \times \rho_{ij} \times \delta_{ij}(\tau_{ij}, m) \times \sigma_{ij} \times \alpha_{ij}]]. \tag{5}$$

However, the measure shown above makes it difficult to compare one RMS, operating within one distributed system, to another RMS operating in a different distributed system. To allow this kind of comparison, the equation

$$\frac{\sum_{j=0}^{t-1} \pi(p_j, m) \times [\max_{0 \le i < I_i} [\eta_{ij} \times \rho_{ij} \times \delta_{ij}(\tau_{ij}, m) \times \sigma_{ij} \times \alpha_{ij}]]}{\text{baseline}} \tag{6}$$

normalizes the collective value of the completed tasks by the results from some baseline, which depends on the tasks and underlying distributed system. This will be discussed further in Subsection 4.2.

Recall the goal of the FISC measure is to determine the performance of a schedule (mapping) for tasks in an oversubscribed distributed system by calculating the collective value of the tasks completed. This measure can also be used as a critical part of an objective function of a scheduling heuristic. Components in addition to the FISC measure may be useful in the determining of the objective function. For example, in [29] the objective function that is used includes expected time between when the data request will be satisfied and its deadline (i.e., urgency).

4.2. Baseline

This section provides a baseline for the FISC ratio. The purpose of the baseline in the FISC ratio is to determine how an RMS performs compared to another RMS in a given environment. If the RMS cannot perform much better than this baseline, then a naive algorithm for resource assignment would perform almost as well as the RMS. The baseline builds upon and extends the example given by [29]. The algorithm used to compute the baseline uses the concept of perfect completion. A task is said to achieve perfect completion if there exists available resources, to which it can be assigned, that would allow it to complete with $\eta_{ij} = \delta_j = \sigma_{ij} = \alpha_{ij} = 100\%$ and $\rho = 1$.

A simple algorithm, which assumes knowledge of the expected resources needed by a task to complete, can be used to obtain a baseline. For the results of the obtained baseline to be reproducible within a certain tolerance, an ordering of the tasks is needed.

The algorithm is shown in Figure 2 and it proceeds as follows. First, it assigns an ordering to the tasks according to their priorities, deadlines, and expected execution times where the above criteria are considered in the aforementioned order. For the tasks with the same priority level, the deadline would be used as a tiebreaker. If tasks have same priority level and deadline, the expected execution time would serve as a tiebreaker. Only if tasks have the same priority, deadline, and expected execution time would the ordering be random. Alternatively, additional characteristics of the task could be used for finer ordering. In other problem domains, other parameters could be more appropriate for ordering the tasks. After the ordering, the algorithm determines whether the first task (according to the ordering) can be expected to achieve perfect completion using the available resources. If so, it computes the expected availability of resources after that task has completed, under the assumption that the task uses the first such available resources. It also adds the weighted priority of this task to the baseline, which was initialized to 0. If a task cannot achieve perfect completion, nothing is added to the baseline and the task is not considered again. The same process is repeated for each task, considering them according to the ordering.

```
order tasks by priority, deadline, and expected
execution time
if all are equal, order is random

if task can get $\eta_{ij} = \delta_{ij} = \alpha_{ij} = \sigma_{ij} = 100\%$
and $\rho_{ij} = 1$
    schedule
    add $\pi(p_j, m)$
    update status of resources
else
    no value added
    no resources consumed
```

Figure 2: Baseline algorithm.

4.3. Generalization

The previous subsection describes one instantiation of the FISC ratio. It can be generalized such that the numerator is any function of $\pi(p_j, m)$, η_{ij}, ρ_{ij}, $\delta_j(\tau_{ij}, m)$, σ_{ij}, and α_{ij} (or other factors), and each of these primary factors can be any function of secondary factors (e.g., primary factor σ_{ij} includes an average of $g_{ij}.c$ secondary factors in the security context described in Subsection 3.5). Let P_r be a primary factor where there can be u number of primary factors ($0 \le r \le u-1$) and s_e be a secondary factor where there can be v_r number of secondary factors ($0 \le e \le v_r - 1$). The generalization of FISC ratio can be represented as

$$FISC = \mathrm{f}(P_0, P_1, \ldots, P_{u-1}) / \text{ baseline and} \qquad (7)$$

$$P_r = \mathrm{f}_r(s_0, s_1, \ldots, s_{v_r - 1}),$$

where each s_e is a secondary factor for P_r. Linear or nonlinear weightings of each factor, depending on the importance of the factor considered in a given environment, may be included in all the functions of primary and secondary factors.

The baseline described is one method of normalizing the numerator of the FISC ratio. Other methods for normalizing could be incorporated to compare the performance of different RMSs in a given environment.

5. Examples of FISC Ratio Use

As an example of how the FISC ratio might be applied in practice, consider the following scenario. The Joint Force Air Component Commander (JFACC) staff are preparing an Air Tasking Order (ATO). As the ATO develops, one tool available to the JFACC staff for its evaluation is the Extended Air Defense Simulation (EADSIM) system from US Army Space and Missile Defense Command. EADSIM is a warfare modeling application offering great flexibility in the areas modeled, the capabilities of the platforms simulated, and the method of simulation (deterministic or stochastic) [21].

EADSIM utilizes a wide range of computing resources, depending on the features enabled. For example, the stochastic mode may use approximately 20 times the computing resources as the deterministic mode (based on the number of runs required to obtain a statistically significant number of samples). Of course, results obtained in stochastic mode are likely to be more reliable.

The JFACC planners select two versions of EADSIM, the stochastic mode and the deterministic mode, and submit them, with different preferences, to their RMS for execution. Because this information is urgently needed for combat mission planning, the priority of this request is seven on a scale of ten (ten being highest). The deadline is firm, with the simulation results required within an hour. If received within an hour the results will achieve some fraction of their maximum worth. After that time, they receive 0 value. The stochastic version is preferred because it will produce higher confidence results, but the deterministic simulation may also be useful. The stochastic version is assigned a preference of eight, on a scale of ten, while the deterministic version is assigned a preference of five. Security level in this case is binary. The information must be sent over a secure link. If it is, a version is assigned a security value of 1, if not, it is assigned a security value of 0. If only one of the two versions can be completed, and these are the only ones to choose from, then the stochastic version will be completed because it will give a higher value than the deterministic version. This is a simple case; usually other factors have to be considered (e.g., expected execution time) when scheduling tasks.

An RMS such as MSHN would evaluate the expected resource requirements of each version as well as the ability to complete each version based on the current resource availability. Using this information, the RMS could make a wise decision by maximizing an objective function where the FISC ratio would be a major component. While this example is from a military environment, the FISC ratio can be adapted for other environments as well.

6. Summary and Future Work

The FISC ratio provides a way to quantify the value of the performance received by a set of applications in a distributed system. Thus, it can be used to evaluate the effectiveness of the mapping of a collection of requests to resources done by a scheduler. In addition, it may be used in a simulation mode to analyze the impact of proposed changes to the distributed system. Therefore, the FISC performance measure presented here will help the distributed computing community in the implementation of resource management systems and the analysis and comparison of such systems. Furthermore, the FISC ratio may be used as a critical part of a scheduling heuristic's objective function. A generalization of the ratio is also discussed in this research. Additional issues that may be considered in future research include weighting the relative importance of the π, η, ρ, δ, σ, and α factors, using a non-linear combination of task values to compute the overall measure, and using of negative fractions in the deadline function in case of catastrophic results from a missed deadline could be incorporated.

Acknowledgments: The authors thank B. Beaton, G. Koob, J. Rockmore, E. Chong, M. Jurczyk, I-J. Wang, S. Jones, T. D. Braun, N. Beck, S. Ali, S. Chatterjea, A. Naik, P. Dharwadkar, J. Kresho, and N. Rowe for their valuable comments.

References

[1] Agile Information Control Environment Proposers Information Package, BAA 98-26, http://web-ext2.darpa.mil /iso/aice/aicepip.htm, Sep. 1998.

[2] L. Badger, D. F. Stem, D. L. Sherman, K. M. Walker, and S. A. Haghighat, "Practical domain and type enforcement for Unix," *1995 IEEE Symp. Security and Privacy*, May 1995, pp. 66-77.

[3] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," *1996 IEEE Symp. Security and Privacy*, May 1996, pp. 164-173.

[4] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *17th IEEE Symp. on Reliable Distributed Systems*, Oct. 1998, pp. 330-335.

[5] S. Chatterjee, B. Sabata, and J. Sydir, *ERDoS QoS Architecture*, ITAD-1667-TR-98-075, SRI, Menlo Park, CA, May 1998.

[6] M. Condell, C. Lynn, and J. Zao, "Security policy specification language," *INTERNET-DRAFT*, Network Working Group, Oct. 1998, ftp://ftp.ietf.org/internetdrafts/ draft-ietf-ipsec-spsl-00.txt.

[7] DARPA, Battlefield Awareness and Data Disse-mination, Apr. 1999, www.darpa.mil/iso/ badd/.

[8] M. M. Eshaghian, ed., *Heterogeneous Computing*, ArTech House, Norwood, MA, 1996.

[9] I. Foster and C. Kesselman ed., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, CA, 1999.

[10] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: the Management System for Heterogeneous Networks," *8th Heterogeneous Computing Workshop* (*HCW '99*), Apr. 1999, pp. 184-198.

[11] C. E. Irvine and T. Levin, "Toward a taxonomy and costing method for security services," *15th Annual Computer Security Applications Conf.* (*ACSAC '99*), Dec. 1999, to appear.

[12] J. H. Kim and H. S. Shin, "Optimistic priority-based concurrency control protocol for firm real-time database systems," *Information & Software Technology*, Vol. 36, No. 12, Dec. 1994, pp. 707-715.

[13] J. K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. St. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, *A Multi-Dimensional QoS Performance Measure for Distributed Heterogeneous Networks*, ECE School, Purdue, technical report in preparation.

[14] J. P. Kresho, *Quality Network Load Information Improves Performance of Adaptive Applications*, Master's Thesis, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, Sep. 1997 (D. A. Hensgen, advisor).

[15] C. G. Lee, Y. K. Kim, S. H. Son, S. L. Min, and C. S. Kim, "Efficiently supporting hard/soft deadline transactions in real-time database systems," *3rd Int'l Workshop on Real-Time Computing Systems and Applications*, Oct./Nov. 1996, pp. 74-80.

[16] T. Levin and C. Irvine, *Quality of Security Service in a Resource Management System Benefit Function*, Technical Report, Computer Science Dept., Naval Postgraduate School, to appear.

[17] J. P. Li and M. W. Mutka, "Priority based real-time communication for large scale wormhole networks," *8th Int'l Parallel Processing Symp.*, Apr. 1994, pp. 433-438.

[18] M. Maheswaran, T. D. Braun, and H. J. Siegel, "High-performance mixed-machine heterogeneous computing," *6th Euromicro Workshop on Parallel and Distributed Processing*, Jan. 1998, pp. 3-9.

[19] M. Maheswaran, "Quality of service driven resource management algorithms for network computing," *Int'l Conf. on Parallel and Distributed Processing Techniques and Applications* (*PDPTA '99*), June/July 1999, pp. 1090-1096.

[20] D. C. Marinescu, "A protocol for multiple access communication with real-time delivery constraints," *IEEE INFOCOM '90*, June 1990, pp. 1119-1126.

[21] N. W. Porter, *Resources Required for Adaptive C4I Models in a Heterogeneous Computing Environment*, Master's Thesis, Dept. of CS, Naval Postgraduate School, Monterey, CA, to appear (D. A. Hensgen, advisor).

[22] A. J. Rockmore, *BADD Functional Description*, Internal DARPA Memo, Feb. 1996.

[23] T. Ryutov and C. Neuman, "Access control framework for distributed applications," *INTERNET-DRAFT*, CAT Working Group, Nov. 1998, ftp://ftp.ietf.org/internet-drafts/draft-ietf-cat-acc-cntrl-frmw-01.txt.

[24] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence, "Taxonomy for QoS specifications," *3rd Int'l Workshop on Object-Oriented Real-Time Dependable Systems* (*WORDS '97*), Feb. 1997, pp. 100-107.

[25] P. A. Schneck and K. Schwan, "Dynamic authentication for high-performance networked applications," *6th Int'l Workshop on Quality of Service* (*IWQoS '98*), May 1998, pp. 127-136.

[26] B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, E. Huh, "DynBench: a dynamic benchmark suite for distributed real-time systems," in *Parallel and Distributed Processing: IPPS/SPDP Workshops*, J. Rolim et al., eds., Springer, Berlin, Apr. 1999, pp. 1335-1349.

[27] J. A. Stankovic, M. Supri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers, Norwell MA, 1998, pp. 13-22.

[28] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk, "A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment," *7th Heterogeneous Computing Workshop* (*HCW '98*), Mar. 1998, pp. 115-129.

[29] M. D. Theys, M. Tan, N. B. Beck, H. J. Siegel, and M. Jurczyk, *Heuristics and a Mathematical Framework for Scheduling Data Requests in a Distributed Communication Network*, TR-ECE 99-2, ECE School, Purdue, Jan. 1999, 58 pp.

[30] C. Wang and W. A. Wulf, "A framework for security measurement," *The Nat'l Information Systems Security Conf.*, Oct. 1997, pp. 522-533.

[31] W.E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. Mackie-Mason, "Some economics of market-based distributed scheduling," *18th Int'l Conf. on Distributed Computer Systems*, May 1998, PP. 612-621.

Determining the Accuracy Required in Resource Load Prediction to Successfully Support Application Agility *

John Kresho
Debra Hensgen
Taylor Kidd
Geoffrey Xie

Computer Science Department
Naval Postgraduate School
Monterey, CA 93940

Corresponding Author: Debra Hensgen
hensgen@cs.nps.navy.mil
(408)656-4074 (voice)
(408)656-2814 (fax)

April 22, 1998

Abstract

Largely due to the proliferation of the World Wide Web, and interfaces such as Netscape, users expect to have many different types of information immediately available. When they encounter a lengthy delay caused by heavy loads on shared resources, such as networks or servers, users often (manually) adapt by requesting different forms of the same information. As both mobile and agent computing becomes more popular, users will expect their applications to automatically adapt to heavy resource loads by fetching the information in a different form, e.g., text instead of graphics. This paper studies the accuracy with which resource loading information, particularly network loading information, must be known in order for applications to successfully, and with agility, adapt. We determine that under many normal conditions, fairly inaccurate estimates of currently available bandwidth suffice. However, when the system is heavily loaded, some strategies can perform much better with very accurate load estimates. That is, assuming that the adaptive applications have hard deadlines for obtaining the data they request, up to 20% more of them will receive some form of that data on time, if the adaptation strategy has a good estimate of available bandwidth. Additionally, in these situations, applications that have a better estimate of bandwidth can deliver, on average, larger sized messages corresponding to, in many cases, higher quality data. Finally, the accuracy with which the bandwidth must be known varies not only with inter-arrival rate, but also with the adaptation strategy used and the percentage of adaptive applications in the system.

1 Introduction

Most of today's applications do not automatically adapt their resource needs to changing computational environments. For example, an Internet browsing application may attempt to load a 5 MByte web page whether the expected time to obtain that web page is 10 minutes (lightly loaded network) or 1 hour (heavily loaded network). In such an application, the burden is placed on the user to manually stop the loading of the current web page and request, for example, similar information in a different format, perhaps without

*This research was supported by DARPA under contract number E583. Additional support was provided by the Naval Postgraduate School and the Institute for Joint Warfare Analysis.

graphics, in order to obtain the desired information in a reasonable amount of time. In this paper, we investigate support for applications that automatically adapt to resource availability by reacting to estimates of loads on those resources. In particular, we investigate, through simulation, how accurate the predictions of resource loads, specifically loads on the network, must be in order for adaptive clients to obtain good performance.

Using simulation, we examine the performance of three different client adaptation strategies. One is a control strategy wherein the client does not adapt, and the other two adapt by making use of resource loading information of varying quality. Our clients set deadlines, after which the requested information is not needed. We use a mix of adaptive and non-adaptive clients and vary the load they place on the network. In one of our adaptive strategies, all clients, not just the adaptive clients, stop sending information if the information does not arrive before its deadline.

In the body of this paper, we first describe the general problem on which we are working, then identify the specific part of this problem that we concentrate on in this paper. In our related work section, we compare this work to other ongoing work in the field of support for adaptive applications and we describe a particular existing service implementation, known as a Communications Server, that supports adaptation by estimating the current network load. We describe in that section how far the Communications Server's estimates were from the actual network load in our experiments. We use the results from these experiments as input for some of our simulations. We then describe our simulation model. Our model includes our adaptation strategies, our simulation parameters, and our methodology. We then present our results. Finally, after briefly describing our current work on an architecture to support adaptive applications, we summarize our conclusions.

2 The Problem

This paper is the first in a series of papers about the software architecture for our Management System for Heterogeneous Networks(MSHN). MSHN leverages our experience in designing, building, and experimenting with SmartNet, which is a scheduling framework for heterogeneous computing [7]. MSHN is an extensible software architecture that is layered on top of native operating systems and network protocols. MSHN is responsible for determining which large scale resources, such as compute servers and file servers, should be used to execute particular requests, and shepherding the solutions to those requests. It is also responsible for determining which version of an *adaptable application* should be used to respond to a request.

MSHN uses a model of the underlying resources, and the systems that manage those resources, to determine both which resources should be used and which version of an application to execute. It uses estimates of resource requirements, which it also maintains, as well as estimates of resource availability to make its determinations. Unlike SmartNet, it does not assume that it has control over all applications using the resources.

One example of an adaptive application is one that is capable of receiving and displaying data in many different formats. Another example is an application that has many different synchronization formats, different ones of which would be more appropriate under various resource and loading environments. Still another example is an application used by someone who would prefer today's weather forecast, but who would accept yesterday's. We expect adaptive client applications, usually using MSHN's libraries, to query MSHN servers to determine the current status of resources. Adaptive applications would then begin to use an application format that was well-suited to the available resources, but might need to adjust their decision if higher priority applications need the resources.

This paper documents our investigation into one aspect of MSHN. The particular problem that this paper deals with is determining how accurate an estimate of resource availability is necessary in order to implement a successful strategy for adapting. Knowing how accurate an estimate is needed will help us when we refine MSHN's architecture, because it will help us trade off the amount of time required to get better estimates against the performance benefits that applications will receive from such estimates.

3 Relationship to Other Work on Adaptive Systems

There are many systems that are currently exploring adaptivity as a response to resource load changes in the areas of high performance computing, real-time network protocols, and in systems that support mobile computing.

Siegel [1] investigates adaptation by reconfiguring a SPMD application into a SIMD application. Researchers at UCSD are examining general approaches to reconfiguring applications based upon the types and numbers of processors available [2].

Xie [9] has developed a new network architecture, called Burst Scheduling, to guarantee the real-time quality of service needs for multimedia applications when the loads on the networks are continually changing. RLM [11] examines receiver initiated adaptation for network applications that execute in a heterogeneous multicast environment.

Several researchers are investigating support for mobile applications [12], [6], [3]. In fact, the term "agility," used in the title of this paper, is defined by Satyanarayanan when describing the Odyssey system, as the speed and accuracy with which an adaptive system detects and responds to changes in resource availability. There are two major differences between Odyssey and MSHN: (i) Odyssey is aimed at mobile applications whereas MSHN is not; and (ii) Odyssey does not consider supporting applications with hard deadlines, whereas many of MSHN's applications, if not completed by their deadlines, are useless. Additionally, most of these systems, have, as a key component some mechanisms for encapsulating data typing information. Finally, QuO [18] tackles the software engineering problem of building applications that are capable of adapting.

The previous work most closely related to this paper is design work on the JTF-ATD Communications Server. The Communications Server is one of the servers provided by the the Defense Advanced Research Projects Agency's (DARPA) Joint Task Force Advanced Technology Architecture [4, 5]. It is the job of the Communications Server to predict the current network loads. The Communications Server obtains its estimates by actively sending different sized packets over the network and recording the amount of time required for a round trip. Using this timing information, it estimates the average bandwidth that is available, along with the latency for packets sent along the network. When queried, the Communications Server reports the instantaneous reading, without anchoring it through the use of historical data. In addition to placing an additional load on a possibly heavily loaded resource, the Communications Server, then, is also subject to inaccuracies caused by the bursty nature of network traffic. Along this same line, Wolski's Network Weather Service [17, 16] implements a method for monitoring current network use and CPU load, using historical information to predict future use and load. Like the Communications Server, whose accuracy we report on in this paper, also estimate bandwidth availability by sending different sized data sets over the network and recording the time required for the transfer.

Since this paper examines the performance of adaptation policies, as a function of the accuracy of bandwidth predictions, we include the expected accuracy of the Communications Server as one data point in our simulations. In order to mimic the Communications Server accuracy in these simulations, we ran several file transfer experiments. While the Communications Server monitored the bandwidth between two computers, we transferred files and retrieved readings of the Communications Server's predictions. We plotted the difference between the actual average bandwidth and the Communications Server's predicted bandwidth and found it to be very close to an exponential distribution with a mean of 1Mbit, offset by 340Kbits.

4 Our Simulation Model

We built discrete event simulations of communication-intensive applications that exchange information with one another over a fully-connected network. In this section, we enumerate the values that are fixed in our simulations and describe the parameters that we varied. Before enumerating these values and parameters, we define several terms:

Node. A location that contains many computers that generate network traffic, such as a business office.

Client. An application that generates its own messages. There are typically many clients at a single node.

Best Case Latency. The amount of time it would take for a message to arrive if it could use all of the bandwidth on a channel. We will denote the best case latency as L_b.

The assumptions we use below are similar to those used in simulations of an initial Communications Server design performed by Teknowledge Federal Systems [15]. The characteristics of the simulation experiments that we will describe reflect the following scenario. Two sites were used, one afloat and the other ashore. Three classes of slightly modified application clients were considered: a time-critical electronic mail system, a time-critical database client, and a time-critical web client. In each case, if data is not received on time, lives may be at stake. Similarly, two of the situations used, when we generate messages on average every two and every three seconds, we consider to be crisis situations. Such a crisis situation might exist if there is a weather emergency (e.g., a tsunami) in the Pacific and decisions to evacuate civilians must be made correctly and quickly. We are all familiar with the fact that in such emergencies phone lines become inaccessible due to overuse, if not due to weather conditions. The same will soon be true for network resources. It is exactly in these emergencies that we wish to ensure that

- the highest priority information is delivered on time, and that
- the users of the applications receive sufficient information to make wise decisions.

4.1 Characteristics Common to all of our Simulation Experiments

Given the definitions above, our simulation models a network with the following properties.

- Our simple network has two nodes. No routing is therefore needed; all messages are sent over direct connections.
- The connection between the two nodes is full duplex with a throughput of 10 Mbits/second. Half of the network's bandwidth, 5 Mbits/second, is available for each direction.
- Each client using the network receives an equal share of the bandwidth.

During the simulation, non-adaptive clients generate **ordinary messages** according to the inter-arrival distribution associated with that particular simulation. Ordinary messages differ from **adaptable messages** in that ordinary messages are available in only a single format. The adaptable messages are generated using the same inter-arrival distribution. All messages have the following attributes.

- A **priority**, P, that ranges between 0 (high priority) and 9 (low priority). We generate the priority using a uniform distribution.
- The **tolerated latency** is the amount of time that the application is willing to wait for the data to arrive, before it considers it to be late. The **deadline** is derived by adding the current time to the tolerated latency. As might be expected, we set up the experiment such that the higher priority messages have smaller tolerated latencies; that is, the higher priority messages needed to arrive at their destination sooner.

  For (ordinary) messages from non-adaptive applications, we set the tolerated latency to:

  $$\begin{array}{rl} 16 \times L_b & \text{if } P \in \{8,9\} \\ 12 \times L_b & \text{if } P \in \{5,6,7\} \\ 10 \times L_b & \text{if } P \in \{2,3,4\} \\ 7 \times L_b & \text{if } P \in \{0,1\} \end{array}$$

  The tolerated latencies for adaptive messages are chosen from a uniform discrete distribution of $\{L_b$ + 30 minutes, L_b + 60 minutes, L_b + 90 minutes$\}$. As these tolerated latencies ultimately represent the maximum amount of time available to get some required information to a human decision maker, say some commander in the field, they are highly arbitrary. Depending upon the decision being made, the tolerated latency can range anywhere from seconds to days. The distribution used above was selected as "reasonable" given a tactical planning scenario.
- Non-adaptive clients send ordinary messages that have different lengths depending upon their class.
  - Class A messages are exponentially distributed around 1 MByte and are generated 60% of the time;
  - Class B messages are exponentially distributed around 1.4 MBytes and are generated 25% of the time; and
  - Class C messages are exponentially distributed around 50 MBytes and are generated 15% of the time.

Finally, we ran each experiment using 10 different sets of random seeds. In each set, a different seed was used for each of the following distributions:

- the inter-arrival rate,
- the message class type generation,
- the distribution that determined whether a client was adaptive or non-adaptive,
- each different message size distribution, and
- the priority.

Also, we verified that, in every case, we ran our simulation long enough to reach a stable state. This is especially important because in some of our experiments, the network was very heavily loaded due to a small average inter-arrival time.

4.2 Adaptation Strategies

In different simulations, we varied the percentage of *adaptive clients* between 1.25% and 100% of all clients. Non-adaptive clients exchange data that is available only in a single format. On the other hand, all data that an adaptive client needs to send is available in any of five formats. The actual sizes for each of the five data formats are chosen from exponential distributions with means 3000 MBytes (8 minutes of color video), 300 MBytes, 30 MBytes, 3 MBytes, and .3 MBytes (simple text description). We assume that our adaptive clients' preference for the various formats decreases with size. That is, the most important format for each client to exchange is the largest one and the smallest format is of least importance.

We ran our simulations using three different client adaptation strategies. In the first strategy, **Strategy 1**, the client first requests a performance prediction from the server; that is, it requests that the server respond with its current estimate of the available bandwidth of the network. The client[1] then calculates whether, based on this predicted bandwidth, it should be able to transmit the largest size format of the required data. If the calculation indicates that this format can be transmitted in its entirety before its deadline, the client begins to send the data. If the client's calculation indicates that this transmission cannot be completed before the deadline, the client iterates through the various size formats from larger to smaller, until it determines the largest one that the client can expect to send in its entirety prior to that deadline, and begins to send it. Periodically the server updates the client with new estimates of available bandwidth. If, based upon a new estimate, the client calculates that it cannot complete sending the format that it is currently transmitting prior to its deadline, it stops transmitting that format and searches for a smaller format that it can expect to complete prior to the deadline and begins transmitting that format. [2]

Strategy 2 is very similar to Strategy 1. The only difference is that in Strategy 2, both adaptive and non-adaptive clients take action if their deadline arrives and they have not completed transmitting their current format; they stop transmitting when the deadline arrives. In Strategy 1, such "late" formats were sent to completion, despite the deadline having passed. Strategy 1 seems to be a very inconsiderate strategy. However, similar strategies, known as "application centric" strategies, are used in many high performance applications [2].

Strategy 3 acts as a control. In this strategy, the client does not really adapt. It always attempts to send the largest format and continues sending it until that format has been transmitted in its entirety. We note that this strategy, though seemingly more wasteful even than Strategy 1, is the default strategy in most Internet web servers.

4.3 Simulation Parameters

In this section we identify our simulation parameters and how we varied them for different simulations. The majority of our experimental space (Figure 1) is focused on the three parameters that we discuss first.

We ran different simulation experiments for different average inter-arrival rates. In each simulation, the amount of time between client message generation is exponentially distributed around the mean. The means for the different experiments are set at 2 seconds, 3 seconds, 15 seconds, and 60 seconds. Given the amount of data being sent in our experiments, the 2- and 3-second average inter-arrival times

[1] In this paper, we attribute much of the work of adapting to the client application. However, in the architecture that we are currently building most of the adaptation work is to be done via common libraries that are linked with the client. In our architecture, the client simply supplies the list of acceptable formats along with a ranking indicating its preference for each format.

[2] The adaptive clients in our current simulation do not ever start sending a larger format.

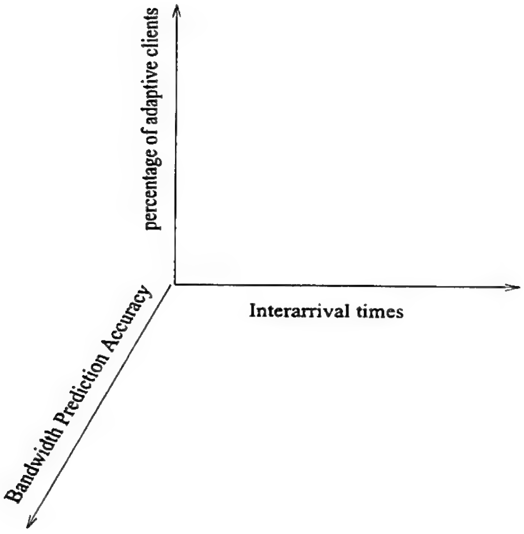


Figure 1: Experimental space using the parameters of inter-arrival times, percentage of adaptive clients, and accuracy of bandwidth estimates.

simulate a critical or busy mode while the 15- and 60-second mean inter-arrival times simulate a lightly loaded mode.

In each experiment, some of our client applications are adaptive, while the remainder are non-adaptive. We ran some experiments where all of the clients (100%) are adaptive and others where very few (1.25%) are adaptive. We also examined some ratios in between these extreems: 20%, 10%, 5%, and 2.5%.

We ran different experiments varying the accuracy with which our server could predict the instantaneously available bandwidth. Since we are running a simulation, instantaneous and exact values of the bandwidth are available. To simulate reality, we added "noise." This noise was sampled from exponential distributions with various different means. Since an exponential distribution produces only positive numbers, we changed the sign of the noise by multiplying it by a sample taken from the discrete uniform distribution {-1,1}, i.e., we flipped a coin. We call this skewed bandwidth estimate the *Instantaneous prediction*. We ran experiments using means, measured in Kbits/second, of 0, 2.5, 5, 7.5, 10, 20, and 50 with the (.85,.15) and (.15,.85) weights described below.

Also, because network traffic is bursty, and because, like all good control systems, we wanted to avoid unstable situations, we had our simulated server use two different sets of **weights** when producing its bandwidth estimation. In the first case it used $Prediction_i = 0.15 \times Instantaneous\ prediction + 0.85 \times Prediction_{i-1}$, where $Prediction_i$ was the estimate used to determine whether the current format could be completed. We also performed experiments using $Prediction_i = 0.85 \times Instantaneous\ prediction + 0.15 \times Prediction_{i-1}$. When we discuss results pertaining to using these different weights in Section 5.2, we will denote them as the weights (x, y), where x refers to the weighing of the $Instantaneousprediction$ and y, to the weighting of the estimate $Prediction_{i-1}$. We note that the Communications Server uses a weighting pair of (1.0, 0.0), which we use for generating data values that correspond to the performance of a server such as the Communications Server.

5 Results

In this section we present results from our simulations and summarize our conclusions from these results. For each experiment, we measured both the average size of the adaptive messages that arrived before their deadline and the percentage of adaptive messages for which no format arrived on time.

5.1 The Need for Adaptation

We first present results that demonstrate the need for adaptation. After these results and our conclusions from them, we restrict our attention to only Strategies 1 and 2 (Section 4.2). As mentioned above, Strategy 3 was our control case wherein adaptive applications simply tried to send the largest format of each data item.

In these experiments, 5% of the processes were adaptive; that is, they attempted to deliver a message of a size drawn from an exponential distribution with mean 3000 MBytes. For all inter-arrival times (means of 2 seconds, 3 seconds, 15 seconds, and 60 seconds), 98% of these large messages did not arrive before their deadline, even when we removed messages immediately if they exceeded their deadlines. In order for these applications to meet critical deadlines, it is apparent that both a method of estimating the network resource load and a strategy for adapting to bandwidth availability is needed.

5.2 The Effect of Varying Weights

In this section, we show that using the weight pair (.15, .85) is substantially better than the pair (.85, .15). In later sections we restrict our discussion to experiments involving only the weight pair (.15, .85).

Using Strategy 1, we again ran simulations where 5% of the messages were adaptive. Table 1 shows results from these simulations. The weighting scheme (.15,.85) is much better when the messages have an inter-arrival mean time of 2 and 3 seconds, and slightly better than (.85, .15) for the 15- and 60-second mean inter-arrival times. The difference between the weighting schemes is a matter of reaction time. Using (.85, .15), adaptive applications will tend to react more quickly to an instantaneous reading which can cause resource thrashing, especially in a heavily loaded environment. On the other hand, the (.15, .85) scheme allows adaptive applications to make better informed decisions based on statistics gathered over a period of time. Ideally, one would want to dynamically adjust these weights depending upon the observed behavior of the data being sampled. That is, the weights would dynamically change based upon detection that the underlying statistics of the sample data is changing. Techniques used to build filters that accomplish this dynamic weighting can be found in Singer [13], LeMay and Brogan [10], and Sworder [14]. While a real system would likely use such filters, we chose to keep our simulation simple and therefore we ran the rest of our simulations, excluding those that mimicked the Communications Server, using the (.15,.85) weighting scheme. As mentioned earlier our simulations that mimicked the Communications Server were executed using the (1.0, 0) weighting because that server does not anchor its predictions with historic information.

INTER ARRIVAL TIME (SECS)	SIZE FOR (.85, .15) WEIGHTING (KBYTES)	SIZE FOR (.15, .85) WEIGHTING (KBYTES)
2	761	1271
3	1833	2546
15	45793	45929
60	297269	299379

Table 1: Average size of messages received using different weighting schemes under Strategy 1 (5% applications adaptive).

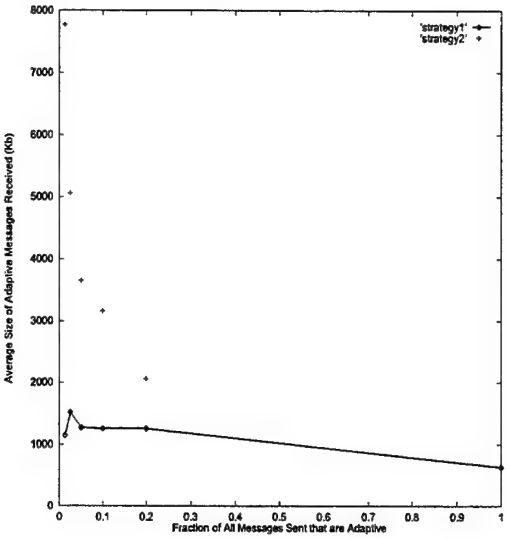


Figure 2: Average size of adaptive messages received for an inter-arrival time of 2 seconds.

5.3 Strategy 1 vs. Strategy 2

In this section, we see that there is some benefit to be gained from dropping messages that exceed their deadline. In comparing these two strategies, we use an *Instantaneous prediction* with a mean error rate of 5.0 Kbits.

Figure 2 shows that when the network resource is very busy (mean inter-arrival time of 2 seconds), the benefit of dropping messages that exceed their deadline is substantial.[3] The results for a 3 second inter-arrival time are very similar.

When the network resource becomes less loaded, there is less benefit from dropping late messages. This is due to the fact that there are fewer messages that are late, and hence eligible to be dropped, because the network resource is not in high demand. Figure 3 shows the results for mean inter-arrival time of 15 seconds. When the mean inter-arrival time is 60 seconds, applications receive sufficient bandwidth the

[3] At this point we note that in Figure 2 which we have already discussed, as well as in both Figure 3 and Figure 4 which we will soon discuss, it may at first appear that performance is decreasing as the fraction of adaptive applications increases. Actually, these graphs must be carefully considered. As the fraction of adaptive applications increases, so does the amount of traffic that the applications collectively attempt to place on the network, since the average size of a message from an adaptive application is larger than the average size of a message from a non-adaptive application.

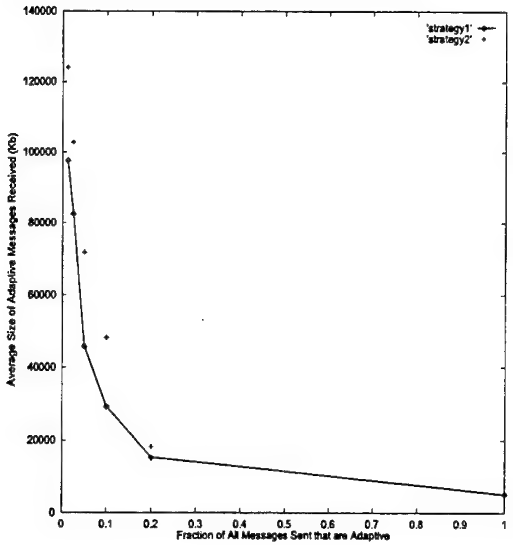


Figure 3: Average size of adaptive messages received for a mean inter-arrival time of 15 seconds.

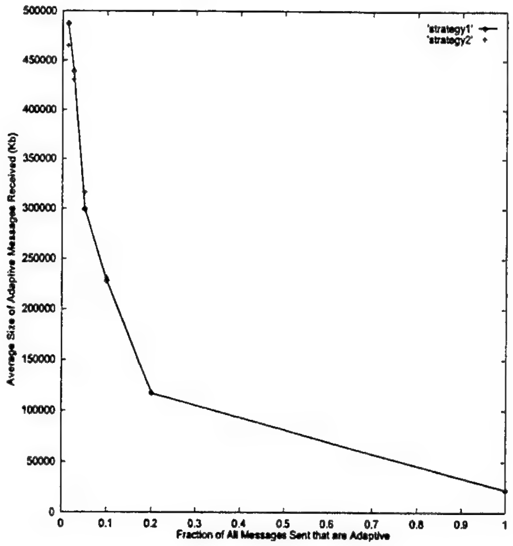


Figure 4: Average size of adaptive messages received for an inter-arrival time of 60 seconds.

majority of the time. Figure 4 demonstrates that in this case there is no benefit derived from, nor penalty paid for, dropping late messages.

5.4 Determining How Accurate Server Estimates Should Be

In this section we examine a multitude of operating points to determine under what conditions:

1. A simple server, such as the JTF-ATD Communications Server, will suffice, and
2. When a more accurate assessment of resource load is needed.

As we stated in the previous section, we performed experiments with the JTF-ATD Communications Server, wherein we recorded the Communications Server's *estimate* of latency based upon its intrusive occasional messages. At the same time as the Communications Server was "snooping" on the network, we ran applications that sent round-trip messages and as accurately as possible measured the *actual* latency and bandwidth. We found that the difference between the estimated and the actual bandwidth, that is, the error in the estimate, very closely approximated an exponential distribution with mean of 1 Mbit, offset 340 Kbits in the negative direction [8]. We therefore used this distribution to predict adaptive performance based upon advice from this intrusive server. The data points in the graph labeled Communications Server correspond to this performance.

Before discussing actual results for different accuracies of server estimates, we note that simulations that use 2 and 3 seconds as their mean message inter-arrival time model a crisis situation. In a military environment, such inter-arrival rates occur in an emergency, such as during a sudden biological attack. In this case, the network resource will be in high demand, but the priority messages must make it to their destinations before their deadlines. However, when the mean inter-arrival times are 15 and 60 seconds, the network resource is under normal use, and not many applications are competing for the same network resource.

In order to determine how accurate a server must be when estimating a resource's load, we collected data for each of the *Instantaneous prediction* error means enumerated above. The first criteria analyzed was the number of messages that did not make their deadlines under Strategies 1 and 2. Figures 5 and 6 display the results of Strategy 2 when there are 100% and 1.25% adaptive messages, respectively. The results showing the number of late messages for Strategy 2 show a trend similar to that of Strategy 1. We note that the points labeled "Communications Server" model the accuracy of the JTF ATD Communications Server as discussed above.

For each of the *Instantaneous prediction* means, including the Communications Server, there were no

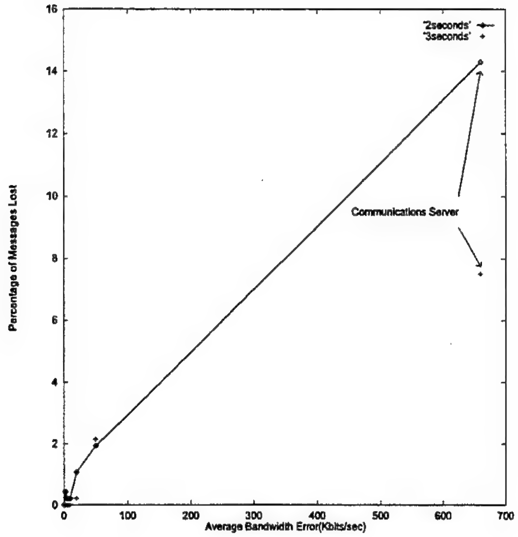


Figure 5: Percentage of messages not received by deadline when using Strategy 2 and 100% of messages are adaptive.

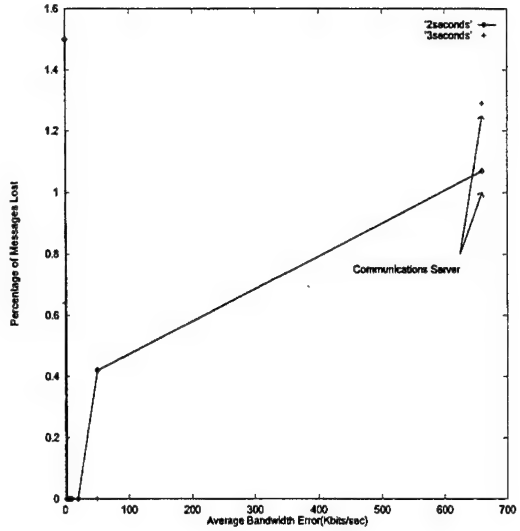


Figure 6: Percentage of messages not received by deadline using Strategy 2 and 1.25% of messages are adaptive.

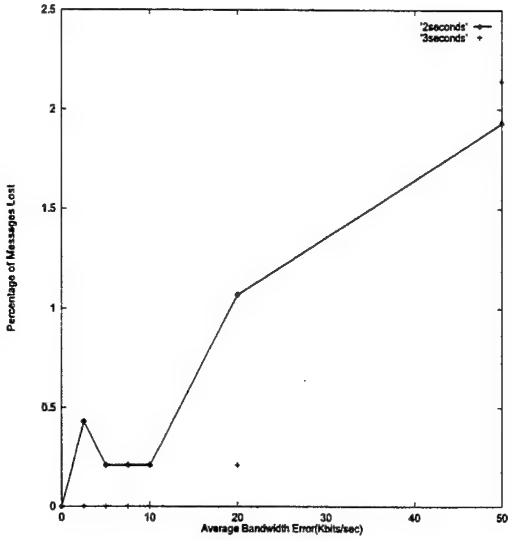


Figure 7: Percentage of messages not received by deadline using Strategy 2 and 100% of messages are adaptive.

late messages for mean inter-arrival times of 15 and 60 seconds. Therefore, we now focus on the crisis situations (2 and 3 second mean inter-arrival times) and determine how accurate a server's prediction must be. Figure 7 and Figure 8 eliminate the Communications Server data and focus on more accurate assessments. Again, the results for Strategy 1 show similar trends. Before continuing further, we wish to point out an interesting phenomena that is seen on close examination of these figures: the small spike at 2.5 KB/sec error in Figure 7 and the downward slope prior to 2.5 KB/sec error in Figure 8. These phenomena result from our applications being too optimistic initially. When many adaptive applications decide almost simultaneously, to send a message, and they have an accurate assessment of the available bandwidth they may all attempt to send very large formats. When they notice that the other applications are also sending large formats, they throttle back, but sometimes it is too late, because they have already put so much data on the network that they prevent one another from completely transmitting any size of message.

After examining these results closely, we determined that, in a crisis situation, being within 20 Kbits/second of the actual network throughput allows most messages to meet their deadlines. Using a less accurate server may result in an unacceptable loss of data, possibly meaning loss of life in some ap-

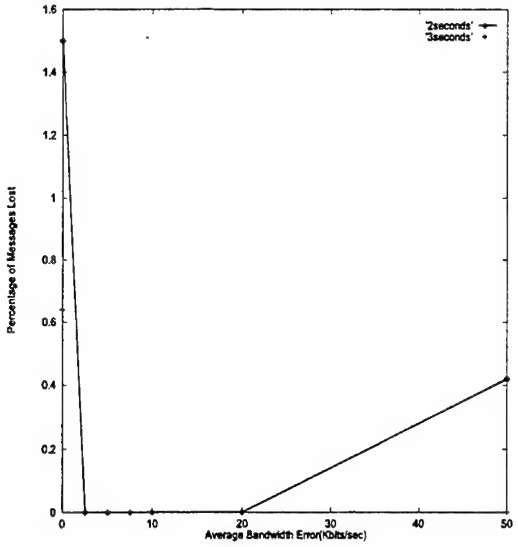


Figure 8: Percentage of messages not received by deadline using Strategy 2 and 1.25% of messages are adaptive.

plications.

The second criteria we examine is the average size of the message that does arrive on time. Figure 9 shows the results for Strategy 2 when 100% of the applications are adaptive. We note that when the server estimates are less accurate, only smaller messages are successfully received. We observed this trend for all inter-arrival times for these simulations.

In order to better understand the circumstances under which the average size received is maximized, we refer to Figure 10. The figure indicates that being within 5 Kbits/second will get the best results in a crisis situation under most loads and being within 10 Kbits/second may suffice in many circumstances. The only exception to this rule is seen when only 1.25% of the applications are adaptive, and the mean inter-arrival rate is 60 seconds. Figure 11 shows that, in this case, a server such as the Communications Server will allow for larger adaptive messages to arrive on time. Since there is little competition for the network resource, a less accurate picture of the load is acceptable. Overall though, as Figure 12 shows, when a crisis situation occurs, it is better to have an accurate server, one that can predict the network bandwidth within 10 Kbits/second. The results for Strategy 1 again are similar to the results presented here.

As can be seen from the results, most cases require an accurate estimate of the network resource

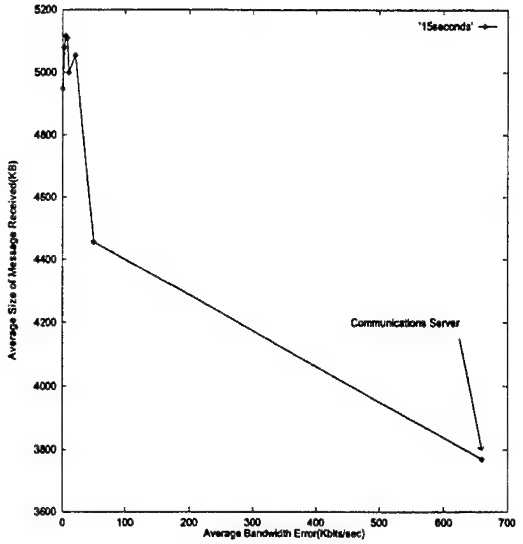


Figure 9: Average size of successful adaptive messages using Strategy 2 when 100% of the messages are adaptive and the mean inter-arrival time is 15 seconds.

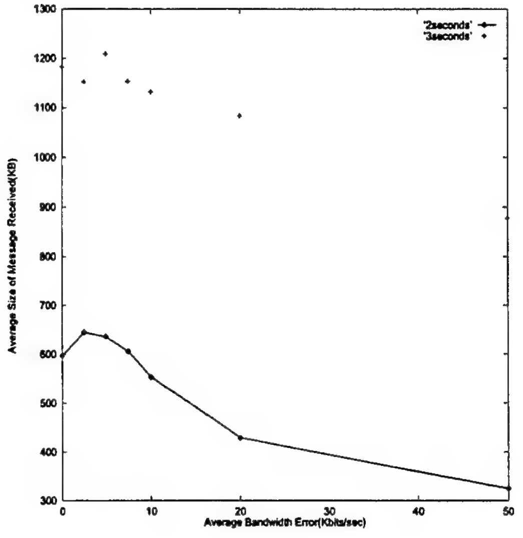


Figure 10: Average size of successful adaptive messages using Strategy 2 when 100% of the messages are adaptive and the mean inter-arrival times are 2 and 3 seconds.

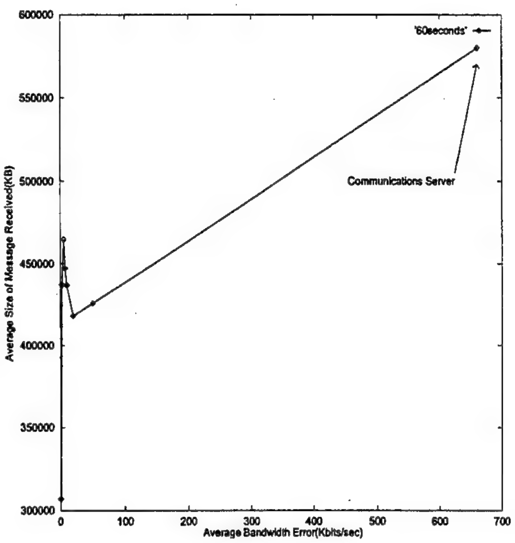


Figure 11: Average size of successful adaptive messages using Strategy 2 when 1.25% of the messages are adaptive and the mean inter-arrival time is 60 seconds.

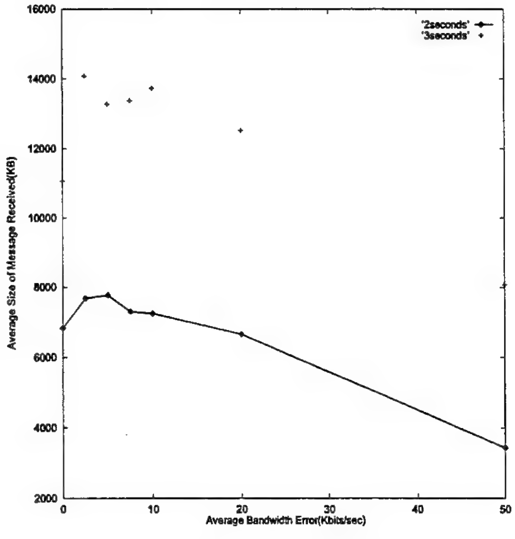


Figure 12: Average size of successful adaptive messages using Strategy 2 when 1.25% of the messages are adaptive and the mean inter-arrival times are 2 and 3 seconds.

(within 10 Kbits/second), especially in a crisis mode. However, when there is little competition for the resource and the percentage of adaptive applications was small, a less accurate estimate may be useful.

6 Current Work

In addition to expanding the portion of the parameter space covered by our simulations for the problem described in this paper, we are currently in the process of refining both the generalized scheduling problem that MSHN addresses as well as our software architecture. The scheduling problem, for which MSHN provides support, is a stochastic pseudo-Boolean programming problem. MSHN's initial architecture is made up of wrappers for system calls; an adaptation client library; two database supporting servers, one maintaining a quickly changing database and another maintaining a fairly rapidly changing database; and a SmartNet-like scheduling advisor. Further description of the MSHN architecture is beyond the scope of this paper and will be documented elsewhere.

7 Conclusions

In this paper we saw that, for situations similar to those examined by Teknowledge, an adaptive client requires a better network resource load assessment than can be furnished by an intrusive server that occasionally examines the state of the network. Specifically, estimating the network resource within 10 Kbits/second allows applications to adapt very well in most cases and in some cases an estimate within 20 Kbits/second suffices. While we make no claim that these results apply for situations with very different parameters than we examined in this paper, we chose these parameters both because they represented realistic scenarios and because they allowed us to compare various adaptation strategies under conditions which had already been examined by others against those same strategies when more information is available.

Finally, this paper only focuses on estimating the network resource load, but other resources, such as CPU and hard drive use, must also be monitored. In order to allow easy development of adaptive applications, an architecture that provides these applications an interface to accurate information, such as that presented in this paper, is essential.

References

[1] Armstrong, J. B., Siegel, H. J., Cohen, W., Tan, M., Dietz, H. G., and Fortes, J. A. B. Dynamic Task Migration from SPMD to SIMD Virtual Machines. In *Proceedings of the International Conference on Parallel Processing* (August 1994), vol. 2, pp. 160–169.

[2] Berman, F., and Wolski, R. Scheduling From the Perspective of the Application. In *High Performance Distributed Computing(HPDC '96)* (August 1996).

[3] Fox, A., Gribble, S. D., Brewer, E. A., Amir, E., Dietz, H. G., and Fortes, J. A. B. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International ACM Conference on Architectural Support for Programming Languages and Operating Systems* (October 1996), pp. 160–170.

[4] Hayes-Roth, F. The JTF-ATD Architecture for Agile, Mobile, Collaborative Crisis Response: A Case Study in Architecture Driven Development. Teknowledge Federal Systems Technical Report, available at http://www.teknowledge.com, March 1995.

[5] Hayes-Roth, F., and Erman, L. Joint Task Force Architecture Specification (JTFAS). Teknowledge Federal Systems Technical Report, available at http://www.teknowledge.com, April 1994.

[6] Inouye, J., Cen, S., Pu, C., and Walpole, J. System Support for Mobile Multimedia Applications. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (May 1997), pp. 143–154.

[7] Kidd, T., Hensgen, D., Freund, R., and Moore, L. Smartnet: A Scheduling Framework for Heterogeneous Computing. *Proceedings of the IEEE International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'96)* (June 1996), 514–521.

[8] Kresho, J. Quality Network Load Information Improves Performance of Adaptive Applications. Naval Postgraduate School Technical Report, Master's Thesis, September 1997.

[9] Lam, S., and Xie, G. Burst Scheduling Networks: Flow Specification and Performance Guarantees. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Video and Audio* (April 1995), pp. 289–292.

[10] LeMay, L. J., and Brogan, W. L. Kalman Filtering: Extended Kalman Filter. St. Joseph Sciences, Inc., 1984.

[11] McCanne, S., Jacobson, V., and Vetterli, M. Receiver-driven Layered Multicast. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM) '96 Conference* (August 1996), pp. 117–130.

[12] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., and Walker, K. R. Agile Application Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles* (October 1997).

[13] Singer, R. A. Estimating Optimal Tracking Filter Performance for Manned Maneuvering Targets. *IEEE Transactions on Aerospace and Electronic Systems AES-6*, 4 (July 1970), 473–483.

[14] Sworder, D. D., Clapp, J., and Kidd, T. Model-Based Prediction of Decisionmaker Delays and Uncertainty. *Cybernetics and Systems 24*, 1 (January 1993), 51–68.

[15] Terry, A., Barnes, T., and Hayes-Roth, R. JTF ATD Communications Server: Architectural Refinement Through Simulation Experiments. Teknowledge Federal Systems Technical Report, available at http://www.teknowledge.com, September 1995.

[16] Wolski, R. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference* (August 1997).

[17] Wolski, R., Spring, N., and Peterson, C. Implementing a performance forecasting system for metacomputing: The network weather service. Tech. Rep. TR-CS97-540, UCSD, May 1997.

[18] Zinky, J. A., Bakken, D. E., and Schantz, R. D. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems 3* (January 1997).

Evaluation of a Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems

YU-KWONG KWOK[1], ANTHONY A. MACIEJEWSKI[2], HOWARD JAY SIEGEL[2],
ARIF GHAFOOR[2], AND ISHFAQ AHMAD[3]

[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong

[2]School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA

[3]Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong

Email: *ykwok@eee.hku.hk, {maciejew, hj, ghafoor}@ecn.purdue.edu, iahmad@cs.ust.hk*

Abstract—To minimize the execution time of an iterative application in a heterogeneous parallel computing environment, an appropriate mapping scheme is needed for matching and scheduling the subtasks of the application onto the processors. When some of the characteristics of the application subtasks are unknown *a priori* and will change from iteration to iteration during execution-time, a *semi-static* methodology can be employed, that starts with an initial mapping but dynamically decides whether to perform a remapping between iterations of the application, by observing the effects of these *dynamic parameters* on the application's execution time. The objective of this study is to implement and evaluate such a semi-static methodology. For analyzing the effectiveness of the proposed scheme, it is compared with two extreme approaches: a completely dynamic approach using a fast mapping heuristic and an ideal approach that uses a genetic algorithm on-line but ignores the time for remapping. Experimental results indicate that the semi-static approach outperforms the dynamic approach and is reasonably close to the ideal but infeasible approach.

1 Introduction

Heterogeneous computing (HC) encompasses a great variety of situations (e.g., see [5], [12]). This study focuses on a particular application domain in which (1) an iterative application is to be mapped onto an associated specific type of dedicated heterogeneous parallel hardware platform and (2) the execution of each iteration can be represented by a directed acyclic graph (DAG) of subtasks. To minimize the execution time of such an iterative application on a heterogeneous parallel computing environment, an appropriate mapping scheme is needed for matching and scheduling of the subtasks onto the processors [12]. However, when some of the characteristics of the application subtasks are unknown *a priori* and will change from iteration to iteration during execution-time, it may not be feasible or desirable to use the same off-line derived mapping throughout the whole execution of the application.

An example of such a problem domain is iterative automatic target recognition (ATR) tasks [14], where a sequence of images is received from a group of sensors and various kinds of operations are required to generate an on-going scene description. In ATR, the characteristics of a subtask's input data, such as the amount of clutter and the number of objects to be identified, may change dynamically and may lead to large variations in the subtask's processing requirements.

In such situations, a semi-static methodology [1], [2] may be employed, that starts with an initial mapping but dynamically decides whether to remap the application with a mapping previously determined off-line. This is done by observing, from one iteration to another, the effects of the changing characteristics of the application's input data, called dynamic parameters, on the application's execution time. Such real-time input-data dependent remapping between iterations can be performed by using an off-line determined mapping. That is, the operating system will be able to make a heuristically-determined decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system will be able to select a pre-computed and stored mapping that is appropriate for the given state of the application. This remapping process will, in general, require a certain system reconfiguration time for relocating the data and program modules.

The application to be mapped is iterative and each iteration is modeled by a DAG in which the nodes represent subtasks and the edges represent the communications among subtasks. The model used for an application task is described in Section 2. The attributes associated with the DAG, such as the computation time of a subtask and the communication time between subtasks, are modeled by equations that are functions of the dynamic parameters. Examples of dynamic parameters include the contrast level of an image, the number of objects in a scene, and the average size of an object in a scene. Thus, as the dynamic parameters change from one iteration (one image) to the next iteration, the mapping currently in use may not be suitable and a remapping of the subtasks onto the processors may need to be performed. However, performing a remapping requires a certain system reconfiguration time. Given the current mapping, a new mapping, and the system estimated reconfiguration time, the operating system has to decide whether a remapping is to be done. This framework can be applied to any task graph structure represented as a DAG.

The objective of this study is to implement and evaluate a semi-static methodology, called the on-line use of off-line derived mappings (denoted as On-Off in subsequent sections), which was originally proposed in [2]. The implementation of the On-Off methodology entails tackling two research issues: (a) how to select representative mappings off-line for on-line use? and (b) is this approach really beneficial compared to a strictly dynamic approach?

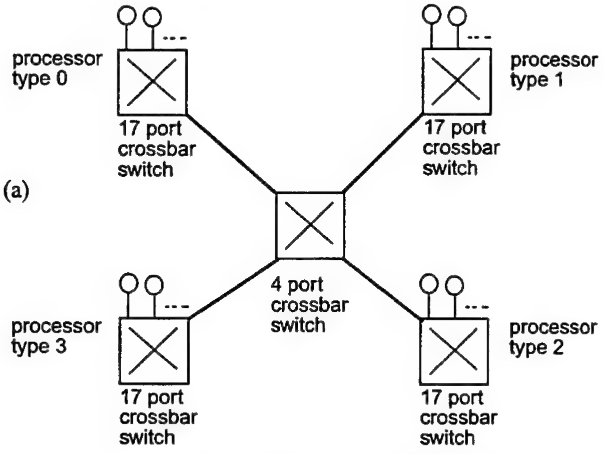
This research was supported by the HKU CRCG, the Hong Kong Research Grants Council (under the contract number HKUST 6076/97E), and the DARPA/ITO Quorum Program (under NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and GSA contract number GS09K99BH0250).

To address these issues, a novel dynamic parameter space partitioning and sampling scheme is proposed in Section 3. During the off-line phase, a genetic algorithm is used to generate high quality mappings for a range of values for the dynamic parameters. During the on-line phase, the actual dynamic parameters are observed and the off-line derived mapping table is referenced to choose the most suitable mapping. Experimental results, presented in Section 4, indicate that this semi-static approach is effective in that it consistently gave performance that was comparable to that of using the same genetic algorithm on-line with the exact dynamic parameter values for the *next* iteration (which is physically impossible). How this research differs from earlier related work, such as [8], [9], [10], [17], and further details about all aspects of this paper are in [7].

2 System Model

To evaluate the On-Off semi-static mapping methodology, a particular sample architecture is chosen; however, the On-Off method can be adopted for other target architectures. The sample target heterogeneous computing platform considered is based on the expected needs of ATR applications that are of interest to the U.S. Army Research Laboratory (e.g., [3]). Specifically, it contains four different types of processors (e.g., SHARC DSP processors and PowerPC RISC processors [4]), with 16 processors of each type (see Figure 1(a)).

The processors are connected via crossbar switches in such a way that each processor has exactly one input port and one output port. Communications among processors of the same type are assumed to be symmetric in the sense that the conflict-free time for any pair of processors (of the same type) to communicate is the same (see Figure 1(b)). For simplicity, it is assumed that if a data-parallel implementation of a given subtask uses a virtual machine of processors, all processors will be of the same type. Given this and the symmetry property of the inter-processor communications among processors of the same type, the expected execution time of a particular multiprocessor implementation of a subtask is independent of which fixed-size subset of the processors of a given type are assigned to execute the subtask.



(b)

	type 0		type 1		type 2		type 3	
	S	*1/R*	*S*	*1/R*	*S*	*1/R*	*S*	*1/R*
type 0	0.92	0.41	9.04	3.72	10.47	5.58	22.47	11.16
type 1	9.04	3.72	0.83	0.37	4.62	0.44	7.00	7.44
type 2	10.47	5.58	4.62	0.44	0.86	0.45	11.37	6.38
type 3	22.47	11.16	7.00	7.44	11.37	6.38	0.83	0.53

Figure 1: (a) The target heterogeneous computing platform consisting of four types of processors with 16 in each type; (b) startup time (*S*) and transmission time per unit data (*1/R*) of the inter-processor communication channels.

An application task is modeled as a DAG, with n nodes representing subtasks s_i $(0 \le i \le n-1)$ and e edges representing inter-subtask communications. To illustrate the efficacy of the On-Off semi-static mapping approach, a simplified model is used for subtask execution time and inter-subtask communication time. However, the On-Off framework does not depend on the form of the equations used and it is the responsibility of the application developer to use an appropriate model [1], [2].

The simple execution time expression used in this model is a version of Amdahl's law extended by a term representing the parallelization overhead (e.g., synchronization and communication). The serial and parallel fractions of a subtask are frequently represented using similar models (e.g., [11]). The execution time expression for subtask s_i includes: (a) three dynamic parameters, α, β, and γ, (b) the number of processors used, p, and (c) three coefficients, a_i, b_i, and c_i [7]. The parallel fraction and serial fraction of subtask s_i are represented by $a_i\alpha/p$ and $c_i\gamma$, respectively. The parallelization overhead is represented by $b_i\beta\log p$ and h_{iu} is the heterogeneity factor, indicating the relative speed of the subtask s_i on the type of processor used in virtual machine *u*. The execution time $t_u(s_i)$ of subtask s_i on virtual machine *u* is modeled by the expression:

$$t_u(s_i) = h_{iu}(a_i\alpha/p + b_i\beta\log p + c_i\gamma)$$

By differentiating this equation and equating it to zero, the optimal value of *p* that leads to the minimum execution time for a given subtask is $p_{\text{opt}} = (a_i\alpha)/(b_i\beta)$. The mapping heuristic will not assign more processors than a subtask's p_{opt}.

It is assumed that the size of the data to be transferred between two subtasks s_i and s_j consists of a fixed portion modeled by a constant d_{ij} (independent of the input) and a variable portion modeled by the product of a coefficient e_{ij} and a dynamic parameter μ. For communication between virtual machines *u* and *v*, S_{uv} and R_{uv} are the message start-up time and the data transmission rate, respectively (see Figure 1(b) for values of S_{uv} and R_{uv} based on [3], [4]). Thus, the inter-subtask communication between subtask s_i on virtual machine *u* and subtask s_j on virtual machine *v* is C_{uv} which is given by: $C_{uv}(s_i, s_j) = S_{uv} + (d_{ij} + e_{ij}\mu)/R_{uv}$.

3 The Semi-Static Mapping Approach

Consider two approaches for remapping application tasks to processors during execution time (between iterations through the DAG):

- **dynamic mapping:** Based on the current values of dynamic parameters, compute a new mapping in real time using a low complexity algorithm.
- **on-line use of off-line derived mappings:** For each dynamic

parameter, some representative values are chosen so that a number of possible scenarios are generated. Using an off-line (i.e., static) heuristic, high-quality mappings for the scenarios are precomputed and stored in a table. During execution of the application, the mapping corresponding to the scenario with values of dynamic parameters *closest to* the actual values is selected from the table to be a possible new mapping [5].

Because a static mapping heuristic (e.g., the genetic algorithm used in this study) can potentially generate solutions of much higher quality than a dynamic mapping algorithm, it is interesting to investigate how well the approach of on-line use of off-line derived mappings (using the genetic algorithm) performs. Notice that even off-line generation of optimal mappings is infeasible because the heterogeneous mapping problem is NP-complete [6] and, thus, exponential time is needed for finding optimal solutions.

In the On-Off semi-static mapping approach, it is assumed that the ranges of the dynamic parameters are known, and are partitioned into a certain number of disjoint regions. Formally, suppose the minima and total range sizes of the dynamic parameters are given by $\alpha_{\min}$, $\beta_{\min}$, $\gamma_{\min}$, $\mu_{\min}$, ζ_α, ζ_β, ζ_γ, and ζ_μ, respectively. The parameter space $\Re$ can be partitioned into K^4 disjoint regions as follows:

$$\Re(i, j, k, l) = \{(\alpha, \beta, \gamma, \mu)\}, \qquad 0 \le i, j, k, l \le K-1$$

where $\alpha_{\min} + i\zeta_\alpha/K \le \alpha < \alpha_{\min} + (i+1)\zeta_\alpha/K$, and the ranges for β, γ, and μ are defined analogously. Here, the parameter space is uniformly partitioned. However, different values of K can be used for each dynamic parameter, depending upon the specifications given by the application developer.

Within each region (defined by specifying values for indices i, j, k, l), N random dynamic parameter vectors are chosen. An off-line heuristic is then applied to determine the mappings for these sample scenarios represented by different dynamic parameter vectors. For a random sample vector v_x ($0 \le x \le N-1$), denote the corresponding mapping by M_x. The mapping for each sample scenario is exhaustively evaluated for every other sample scenario in the region by applying the mapping to the DAG and computing the completion time. That is, the task execution times $t(M_x(v_y))$ for all x and y ($0 \le x, y \le N-1$) are computed. The mapping M_x that gives the minimum average completion time $[\sum_{y=0}^{N-1} t(M_x(v_y))]/N$ is chosen as the representative mapping for the corresponding region in the dynamic parameter space. This representative mapping and the corresponding average completion time are stored in the off-line mapping table, which is a multi-dimensional array indexed by i, j, k, l.

The input to the simulated on-line module consists of an execution profile that comprises a certain number of iterations of executing the task graph. Examples of execution profiles containing 20 iterations are shown in Table 1 in Section 4. In each profile, the dynamic parameter values change from one iteration to another. Specifically, row i represents the values of the dynamic parameters *observed* after execution of the graph for iteration i is finished. Thus, when execution of the task for iteration i begins, the on-line module does not know the (simulated) *actual* values of the dynamic parameters for that iteration. The on-line module has to determine a mapping for iteration i based on the dynamic parameter values of iteration $i-1$. In the On-Off semi-static mapping approach, given the dynamic parameter values of iteration $i-1$, the on-line module retrieves the mapping corresponding to the representative dynamic parameter vector *closest to* the given actual values. If the stored pre-determined execution time of the selected mapping, plus the estimated reconfiguration time, is smaller than the (simulated) actual execution time of iteration $i-1$, a remapping is performed; otherwise, the mapping used at iteration $i-1$ will continue to be used for iteration i.

In this study, several mapping approaches are examined. The first approach is a dynamic approach that uses a fast heuristic that takes a small amount of time but generates a reasonably good solution. The heuristic used is a fast static scheduling algorithm, called the Earliest Completion Time (ECT) algorithm, that is based on the technique presented in [15].

The method of using the ECT algorithm for scheduling the dynamic iteration tasks is as follows. The ECT algorithm is applied (in real time) to the task graph with the values for the dynamic parameters at iteration $i-1$. The resulting mapping (with its associated estimated task execution time using the iteration $i-1$ parameters) is then considered to be a potential new mapping for iteration i. Again, if the gain in adopting the new mapping is greater than the reconfiguration time, the new mapping will be used in iteration i.

Genetic algorithms (GAs) are a promising heuristic approach to optimization problems that are intractable. There are a great variety of approaches to GAs (see [13] for a survey). The details of our particular GA are available in [7], [16]. However, it should be noted that the On-Off approach does not rely on GAs per se and can be used with any global optimization scheme.

4 Performance Results

Four approaches were compared in the experiments: (i) the On-Off approach; (ii) the ECT algorithm as a dynamic scheduling algorithm; (iii) the infeasible approach of using the GA as a dynamic scheduling algorithm (referred to as GA On-line); and (iv) an ideal but impossible approach which uses the GA on-line with the exact (as yet unknown) dynamic parameters for the iteration to be executed (referred to as Ideal).

To investigate the performance of the On-Off approach with the proposed dynamic parameter space partitioning and sampling methods, task graphs with four different structures were used. These graphs included in-tree graphs, out-tree graphs, fork-join graphs, and randomly structured graphs.

Graphs with sizes 10, 50, 100, and 200 nodes were considered. For each graph structure and size, ten graphs were used in the simulation studies. Thus, a total of $4 \times 4 \times 10 = 160$ different graphs were generated.

In each graph, the coefficients of the subtask execution time equation (a_i, b_i, and c_i) and inter-subtask communication time equation (d_{ij} and e_{ij}) were randomly generated from uniform distributions with ranges [10..100] and [1..10], respectively. The heterogeneity factors of these graphs were also randomly selected from a uniform distribution with range 0.5 to 20.

Details of random task graph generation are given in [7].

The heterogeneous platform shown in Figure 1 was used throughout the experiments. Below are the parameters used in the experiments, unless otherwise stated.

- ranges of the dynamic parameters: α : [1,000..5,000], β : [5..25], γ : [100..500], and μ : [20..100]
- partitioning of the dynamic parameter space: the range of each dynamic parameter is partitioned into four equal intervals (i.e., $K = 4$) and, therefore, the mapping table stores $4^4 = 256$ mappings (i.e., there are 256 regions in the four dimensional space)
- number of randomly chosen sample scenarios within each partition (hyper-rectangle) of the dynamic parameter space: ten (i.e., $N = 10$)
- the GA is executed ten times for each sample scenario, from which the best mapping is chosen (thus, for a single graph, the GA was executed a total of $K^4 \times N \times 10 = 256 \times 10 \times 10$ = 25,600 times to build the mapping table)
- estimated reconfiguration time: 1,000 (calculation of the estimated reconfiguration time is discussed in [2])
- crossover and mutation probabilities for the GA: both 0.4 (these values were chosen according to the light/moderate load results in [16])

Two randomly generated 20-iteration execution profiles of dynamic parameters were used for each graph (see Table 1). The dynamic parameters for Profile B change eight times more rapidly, on average, than those for Profile A. In the profiles, iteration 0 is the initialization iteration. In this study, the dynamic parameter values of iteration 0 are chosen to be the mean values of the respective dynamic parameter ranges. The dynamic parameter values shown on iteration i ($i > 0$) simulate the actual dynamic parameter values *observed* after the task graph finishes iteration i execution. Both Profile A and Profile B are generated randomly based on a single parameter: the mean percentage change in dynamic parameter values, called Δ. Specifically, given Δ, an increment factor, δ_{i-1}, was randomly chosen from a uniform distribution with a range [0.5Δ, 1.5Δ], and its sign had a 0.5 probability of being positive (but selected to guarantee that the dynamic parameter stays within its range). Then, a dynamic parameter for iteration i, say μ_i, was given by: $\mu_i = \mu_{i-1} \pm \delta_{i-1}\mu_{i-1}$.

To examine the performance of the On-Off approach, first consider the results of scheduling a ten-node random task graph using the two execution profiles. The parameters of the ten-node random task graph are shown in Figure 2. Detailed results of using the four approaches for Profile B are shown in Table 2. Due to space limitations, results for Profile A are not shown here but can be found in [7]. Below are the definitions of the data columns.

- **t(M[i–1]):** this is the task execution time of iteration i using the mapping ***chosen*** at the end of iteration $i-1$, denoted by M[i–1]. Here, it should be noted that at the end of iteration $i-1$, a new mapping will be determined but such a mapping would not be used for iteration i if the reconfiguration time offsets the gain of remapping. Thus, a mapping chosen at the end of iteration $i-1$ could be a new mapping or the same mapping used for iteration $i-1$. In the case of On-Off, the new mapping considered is the mapping corresponding to the scenario closest to the parameters at iteration $i-1$ in the mapping table. In the case of ECT, the new mapping considered is the one determined using the ECT algorithm with the exact dynamic parameters at iteration $i-1$.

Table 1: Execution profiles of dynamic parameters: Profile A (average percentage change in dynamic parameter values Δ = 5%) and Profile B (Δ = 40%).

iteration	α	β	γ	μ
0	3000	15	300	60
1	2821	15	287	63
2	2949	12	302	65
3	3073	12	286	68
4	3228	11	273	71
5	3090	13	258	67
6	3256	11	272	70
7	3424	16	259	73
8	3621	16	271	75
9	3811	13	260	78
10	4014	17	245	81
11	4229	13	257	77
12	3994	19	242	80
13	4179	15	253	83
14	4386	15	264	78
15	4208	13	249	82
16	4016	14	236	77
17	3835	16	226	81
18	4026	19	238	84
19	4258	16	251	88
20	4479	15	265	92

(a) Profile A

iteration	α	β	γ	μ
0	3000	15	300	60
1	4309	15	409	82
2	2635	7	268	43
3	3894	6	361	27
4	2241	8	197	39
5	1265	12	287	52
6	1699	16	420	75
7	1138	11	282	50
8	1543	12	153	67
9	2205	17	225	97
10	3198	10	332	51
11	4678	18	477	73
12	2588	8	315	48
13	1358	16	211	67
14	1794	17	307	98
15	2605	11	163	61
16	3719	17	240	87
17	2478	9	332	53
18	1507	16	466	76
19	2081	8	243	50
20	3053	17	149	70

(b) Profile B

- **t'(O[i–1]):** this is the task execution time of the mapping stored in the off-line mapping table, denoted by O[i–1], of a scenario with dynamic parameters closest to the dynamic parameters at iteration $i-1$. This is the value stored in the mapping table, instead of the task execution time by applying O[i–1] to the dynamic parameters at iteration $i-1$. The mapping O[i–1] may or may not be chosen for iteration i.
- **RC:** the reconfiguration time, if remapping is performed.
- **t(E[i–1]):** this is the execution time of the mapping determined using the ECT algorithm with the parameters at iteration $i-1$, denoted by E[i-1], which may or may not be chosen for iteration i.
- **t(G[i–1]):** this is the task execution time of iteration i by applying the mapping determined by the GA with dynamic parameters at iteration $i-1$, disregarding whether the remapping is justified by the gain or not. Also, the mapping found at iteration $i-1$ is incorporated in the initial population of the GA at iteration i. Reconfiguration time is not counted in this approach. Again, this GA On-line method is included for comparison only because applying the GA on-line for dynamic scheduling is not feasible due to the long execution time required by the GA. It should be noted that for both Ideal and GA On-line, reconfiguration time is not considered.
- **t(G[i]):** this is the task execution time of iteration i determined using the GA with the exact dynamic parameters at iteration i. This is, therefore, the ideal case which is

(a)

	coefficients			heterogeneity factors			
	a_i	b_i	c_i	h_{i0}	h_{i1}	h_{i2}	h_{i3}
s_0	9	24	49	0.4897	0.6815	0.7711	0.7503
s_1	42	61	9	0.2828	0.7129	0.4511	0.2725
s_2	42	50	43	0.2575	0.8511	0.5096	0.8779
s_3	2	34	47	0.6337	0.6921	0.8479	0.3451
s_4	9	10	38	0.8283	0.2745	0.4114	0.2836
s_5	33	59	76	0.7267	0.3124	0.2600	0.3354
s_6	63	45	29	0.3932	0.7026	0.8072	0.8066
s_7	56	14	54	0.5276	0.7990	0.5081	0.7942
s_8	48	68	10	0.5876	0.3863	0.6515	0.6472
s_9	36	25	29	0.8760	0.8794	0.6965	0.2407

(b)

	d_{ij}	e_{ij}
s_0->s_1	5	3
s_0->s_2	6	6
s_0->s_3	2	1
s_0->s_4	7	5
s_0->s_5	3	7
s_0->s_8	5	6
s_1->s_7	3	2
s_2->s_9	7	7
s_4->s_6	4	2
s_6->s_7	5	6
s_6->s_8	7	8
s_6->s_9	9	10

Figure 2: (a) Coefficients of the subtask execution time equation and heterogeneity factors h_{iu} for the subtask execution times; (b) coefficients of the inter-subtask communication data equations.

impossible in practice because the actual values of the dynamic parameters for iteration i cannot be known before the execution of iteration i begins. Furthermore, the solutions found by both the On-Off and the GA On-line approaches are also incorporated into the initial population of the GA. This is done in order to determine the "best" solution as a reference for comparison.

As can be seen from Table 2, the On-Off approach of dynamically using off-line derived mappings generated much smaller total execution time (995,987) compared to that of using the ECT algorithm (1,447,666). The On-Off approach consistently resulted in performance comparable to the infeasible GA On-line scheme and was only marginally outperformed by the Ideal (but impossible) method. Indeed, one very interesting observation is that at some iterations, the On-Off approach generated shorter mapping execution time than the GA On-line due to the On-Off approach being more robust to changes.

Table 2: Results for the ten-node random graph using Profile B ("total" below is the task execution time for 20 iterations).

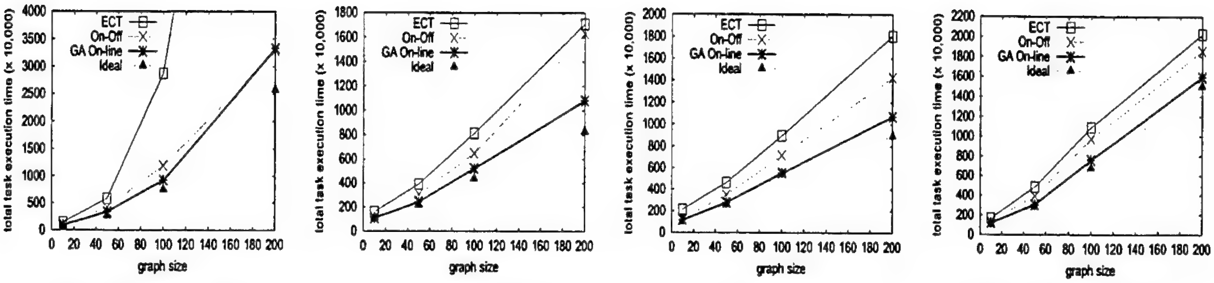
	On-Off			ECT			GA On-line	Ideal
i	t(M[i–1])	t'(O[i–1])	RC	t(M[i–1])	t(E[i–1])	RC	t(G[i–1])	t(G[i])
0	—	54391	1000	—	76029	1000	—	47980
1	61988	69217	0	107754	103630	1000	66150	61988
2	36411	38707	0	63998	61852	1000	45692	34972
3	49142	45126	1000	72559	79625	0	44712	40318
4	43516	26908	1000	51686	51648	0	38337	28418
5	38740	34726	1000	64508	49440	1000	35719	32825
6	50402	48248	1000	74503	74365	0	44250	29231
7	34390	50402	0	50107	49296	0	29842	27753
8	41947	33684	1000	62863	45772	1000	48713	31601
9	45940	48911	0	83088	85888	0	48095	44184
10	57662	48178	1000	75503	76404	0	55606	42139
11	64180	62401	1000	111327	112449	0	69058	61897
12	39930	42374	0	67829	68907	0	40476	39930
13	46949	40904	1000	59054	47534	1000	46350	31203
14	46677	45483	1000	84529	67120	1000	45041	41910
15	58622	41498	1000	54991	56205	0	34473	26400
16	61211	55142	1000	78358	80091	0	57254	51032
17	46193	42374	1000	69973	72405	0	46243	36274
18	56042	41947	1000	85659	72229	1000	55192	44846
19	49145	38707	1000	52644	57459	0	47570	31678
20	51900	—	—	68733	63337	—	48931	43881
total:	995,987			1,447,666			947,704	830,460

Some additional experiments using larger graphs were also conducted to further test the effectiveness of the sampling strategy used. Specifically, ten 50-node random graphs were used and the following different approaches to generate representative mappings were compared:

- the mid-point of the hyper-rectangle is chosen as the only sample scenario for generating the representative mapping (called Scheme 1);
- a randomly selected point within the hyper-rectangle is chosen as the only sample scenario (called Scheme 2);
- ten sample scenarios in the hyper-rectangle are examined but for each sample scenario the GA is executed without incorporating the solution generated by the ECT algorithm as one of the members in the initial population (and the mappings of the ten sample scenarios are applied to all other sample scenarios in the hyper-rectangle, with the mapping giving the best average performance selected) (called Scheme 3);
- the approach used throughout all previous experiments—like Scheme 3 except the GA is executed with the ECT solution as one of the seed chromosomes (called Scheme 4).

The above four approaches were applied to the ten 50-node random graphs using Profile A and the total mapping execution times were noted. The averages of these execution times were determined and the results were normalized with respect to those of Scheme 4. The results obtained are as follows: 1.23 (Scheme 1), 1.19 (Scheme 2), 1.16 (Scheme 3), and 1.00 (Scheme 4). Thus, using just one sample scenario is not as effective as using ten. The degradations are indeed quite significant. Furthermore, the degradation is higher if the mid-point instead of a randomly selected point within the hyper-rectangle is used. These results lead to the conclusion that the relationship between the parameters space and the mappings space is highly irregular and, as such, more random sample scenarios are needed to more accurately "characterize" a good representative mapping for a hyper-rectangle. Finally, as expected, the solutions of the GA without using mappings determined by ECT are worse. Given these findings, Scheme 4 was used throughout all subsequent experiments.

To investigate the effects of graph sizes and structures on the performance of the On-Off approach, the experiments were repeated for larger task graphs. As can be seen from Figure 3,



(a) random task graphs (b) in-tree task graphs (c) out-tree task graphs (d) fork-join task graphs

Figure 3: Total execution times for Profile B.

the total execution times of On-Off, GA On-line, and Ideal for totally random task graphs are consistently of similar values for all graph sizes. However, the ECT approach performed much worse, especially for large graphs with sizes 100 and 200 (in order to enhance the readability of the plots, the data for 200-node graphs of the ECT algorithm were excluded). An explanation for this phenomenon is that because the ECT algorithm employs a strictly greedy scheduling method, the effect of making mistakes at early stages of scheduling can propagate until the whole graph is completely scheduled. The adverse impact of such a greedy approach can be more profound for larger graphs. For other regular graphs, in most cases the performance of On-Off was only slightly inferior to the GA On-line, which in turn was slightly outperformed by the Ideal.

An experiment was also conducted to explore the effect of increasing the reconfiguration time on the performance of the On-Off approach. The total execution times of the ECT and On-Off approaches remained approximately constant despite the fact that the reconfiguration time varied over a wide range (the GA On-line and Ideal results are independent of reconfiguration time). This is due to the observation that a higher reconfiguration time simply prohibited the attempts to switch to a new mapping from one iteration to another. If reconfiguration cost is small, more remappings are performed but the aggregate reconfiguration costs do not considerably affect the total time.

5 Conclusions

A novel strategy is presented for partitioning and sampling the dynamic parameter space of a heterogeneous application within the context of a semi-static mapping methodology. A summary of the extensive performance evaluation of this strategy is given (details in [7]). Experimental results indicate that the semi-static approach is effective in that it consistently outperformed a fast dynamic mapping heuristic, and gave reasonable performance compared with the infeasible approach of directly using the genetic algorithm on-line for a wide range of task graph structures. A limitation of such a semi-static approach is the additional off-line execution time needed to build the mapping table. However, because the mapping table is built off-line and the target heterogeneous task graph is used as a production job, some extra time is affordable.

Acknowledgments—The authors thank M. Maheswaran and T.D. Braun for their helpful suggestions.

References

[1] J.R. Budenske, R.S. Ramanujan, and H.J. Siegel, "Modeling ATR Applications for Intelligent Execution upon a Heterogeneous Computing Platform," *Proc. IEEE Int'l Conf. Sys., Man, and Cyb.*, pp. 649-656, Oct. 1997.

[2] —, "A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms," *J. Supercomputing*, vol. 12, no. 4, pp. 387-406, Oct. 1998.

[3] P. David, P. Emmerman, and S. Ho, "A Scalable Architecture System for Automatic Target Recognition," *Proc. 13th AIAA/ IEEE Digital Avionics Sys. Conf.*, pp. 414-420, Oct. 1994.

[4] T.H. Einstein, "Mercury Computer Systems' Modular Heterogeneous RACE Multicomputer," *Proc. 6th Heterogeneous Computing Workshop*, pp. 60-71, Apr. 1997.

[5] M.M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

[7] Y.-K. Kwok, A.A. Maciejewski, H.J. Siegel, A. Ghafoor, and I. Ahmad, "Implementation and Performance Study of a Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems," Technical Report HKUST-CS99-15, Department of Computer Science, HKUST, April 1999.

[8] C. Lee, Y.-F. Wang, and T. Yang, "Global Optimization for Mapping Parallel Image Processing Tasks on Distributed Memory Machines," *J. Parallel and Distributed Computing*, vol. 45, no. 1, pp. 29-45, Aug. 1997.

[9] D.M. Nicol and J.H. Saltz, "Dynamic Remapping of Parallel Computations with Varying Resource Demands," *IEEE Trans. Computers*, vol. 37, no. 9, pp. 1073-1087, Sept. 1988.

[10] S. Ramaswamy, S. Sapatnekar, and P. Banerjee, "A Framework for Exploiting Task and Data Parallelism on Distributed Memory Multicomputers," *IEEE Trans. Parallel and Distributed Sys.*, vol. 8, no. 11, pp. 1098-1116, Nov. 1997.

[11] K.C. Sevcik, "Characterizations of Parallelism in Applications and Their Use in Scheduling," *Performance Evaluation Rev.*, vol. 17, no. 1, pp. 171-180, May 1989.

[12] H.J. Siegel, J.K. Antonio, R.C. Metzger, M. Tan, and Y.A. Li, "Heterogeneous Computing," in *Parallel and Distributed Computing Handbook*, ed. A.Y. Zomaya, McGraw-Hill, New York, NY, pp. 725-761, 1996.

[13] M. Srinivas and L.M. Patnaik, "Genetic Algorithms: A Survey," *Computer*, vol. 27, no. 6, pp. 17-26, June 1994.

[14] J.G. Verly and R.L Delanoy, "Model-Based Automatic Target Recognition (ATR) System for Forwardlooking Groundbased and Airborne Imaging Laser Radars (LADAR)," *Proc. IEEE*, vol. 84, no. 2, pp. 126-163, Feb. 1996.

[15] Q. Wang and K.H. Cheng, "List Scheduling of Parallel Tasks," *Inf. Proc. Lett.*, vol. 37, no. 5, pp. 291-297, Mar. 1991.

[16] L. Wang, H.J. Siegel, V.P. Roychowdhury, and A.A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *J. Parallel and Distributed Computing*, vol. 47, no. 1, pp. 8-22, Nov. 1997.

[17] T. Yang and C. Fu, "Heuristic Algorithms for Scheduling Iterative Task Computations on Distributed Memory Machines," *IEEE Trans. Parallel and Distributed Sys.*, vol. 8, no. 6, pp. 608-622, June 1997.

An Approach to Characterizing Resource Usage and User Preferences in Benefit Functions[1]

Tim Levin
Anteon Corporation
Monterey, CA

Cynthia Irvine
Naval Postgraduate School
Monterey, CA

Abstract. *An approach is described for representing the level of resources consumed by jobs under the control of a Resource Management System, and it is shown how this measurement of resource usage can be combined with a notion of user preferences to reflect a restrictive resource-usage policy for network management.*

1 Introduction

Various mechanisms exist for managing contention and allotment of distributed network resources. One class of these mechanisms attempts to schedule, in the most efficient means possible, the execution of multiple, simultaneous, jobs on multiple distributed, heterogeneous, computers [1] [2] [3] [9] [10] , where each job requires a determinable subset of the resources.

Abstract benefit functions can be used to measure the effectiveness of resource management systems in satisfying various system and user requirements in the operation of a virtual heterogeneous network. Such measurements can be used to drive the RMS scheduling mechanism, as well as to study the behavior of the RMS.

Of interest is a function for comparing the relative benefits of job scheduling mechanisms when they are presented with real or hypothetical "data sets" of jobs. As we are not considering a "benchmark" type of definition which has a predefined data set, the benefit function needs to be fair regarding the nature of the resource consumption attempted. For example, when comparing mechanism "A," scheduling a set of jobs which require 50% of the resources, to mechanism "B," scheduling a set of jobs which require 98% of the resources, the benefit function needs to give more credit for scheduling the more difficult data set (B, in this example).

We develop, in Section 2, an efficiency metric for showing the effectiveness of an RMS in scheduling jobs with respect to resource usage. Usage is represented as a ratio of *resources scheduled* to *resources available*. Then, in Section 3 a benefit function is developed, utilizing the efficiency metrics as well as representations of a job's priority and preference. A conclusion follows in section 4.

2 Resource Usage Efficiency Metric

In the network computing context, users or user programs may request the execution of "jobs," which are scheduled by an underlying control program to execute on local or remote computing resources. The execution of the job may access or consume a variety of network resources, such as: local I/O device bandwidth; internetwork bandwidth; local and remote CPU time; local, inter-

[1] Funded through MSHN, a DARPA/QUORUM project.

mediate (e.g., routing buffers) and remote storage. The resource usages may be temporary or may persistent for the duration of the job. As there are multiple users accessing the same resources, there are naturally various allotment, contention, and security issues associated with the use of those resources.

The jobs in a particular "data set" are represented by the set J. The number of jobs in J is n. To characterize resource usage, we will abstract both time and resource usage. We measure time in *Time Units*. The total amount of a given resource available during a time unit is one *Resource Unit*. Note that this abstract unit will be used to measure the proportion of a resource consumed by a job, rather than the magnitude of a resource in a given time unit. Each job has an associated deadline, before which it must finish. Deadlines are measured in time units, from the data set's start time. Thus, some jobs may be delayed to start later than other jobs, but this does not affect the deadlines of the delayed jobs. The length of the longest deadline in J, is T. This can be understood as the overall deadline for the data set. The number of different resources available is $[R]$. The total number of resource units available over T is:

$T x [R]$

During each Time Unit ($0 <= t <=$ T), a job (j) requires a fractional amount ($0 <= C_{jrt} <= 1$) of each available resource (r). A resource is considered applicable to a job if and only if the job requires some but not more than 100% of the resource. The relationship of time, resources and jobs can be represented in a three-dimensional matrix, as in Table 1 . Here, we see that in time unit 2, job 1 requires 80% of resource 1 and 20% of resource 2. Notice, that there are some resource conflicts between these two jobs. In time unit 2 for resource 1 and time unit 3 for resource 2, the two jobs require more than 100% of the available resources. The RMS could resolve this conflict, for example, by delaying the start of job 2 for one time unit.

Table 1: Example Resources Required per job and time unit

Resources	Jobs	Time Units (T = 5)				
		1	2	3	4	5
Resource 1	job 1	.5	.8	0	.5	.5
	job 2	.5	.5	.8	.2	0
Resource 2	job 1	.2	.2	.5	.2	.2
	job 2	.3	.5	.7	.8	0

The fraction of resources "required" to resources "available" over the whole data set is:

$$K_{ideal} = \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{r=1}^{[R]} C_{jrt}/(T[R])$$

If the numerator of the above expression is greater than the denominator, then the jobs cannot be scheduled within the specified deadlines; otherwise, $0 <= K_{ideal} <= 1$.

Now, a job will succeed in scheduling only a certain fraction (K_{actual}) of the resources that it requires. For example, a job may run in a degraded format. We introduce the variable Q to indicate the actual scheduling of a given resource:

$\forall t \leq T, r \in R, j \in J$ (if resource *r* is scheduled for job *j* in time *t*, then $Q_{jrt} = C_{jrt}$, else $Q_{jrt} = 0$)

The model is that a job either gets all of the resources it requires (C_{jrt}) in a time unit, or none of it. Degradation with respect to required resources may occur over time, but not within a time unit.

K_{actual} is the ratio of resources scheduled to resources required by all jobs:

$$\mathrm{K_{actual}} = \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{r=1}^{[R]} Q_{jru}/C_{jru}$$

$0 <= K_{actual} <= 1$

As stated, the jobs the scheduler attempts to schedule require some fraction (K_{ideal}) of the number of available resources. Intuitively, K_{actual} and K_{ideal} look like this:

K_{actual} = number of scheduled resource units / number of required resource units
K_{ideal} = number of required resource units / number of available resource units

Recall that we wanted to temper the measurement of a mechanism's success at scheduling (viz, K_{actual}) with a notion of how hard of a job it had attempted (viz, K_{ideal}). The efficiency of a network job-scheduling mechanism can be characterized by multiplying the success rate by the difficulty rate:

$$\mathit{Efficiency} = K_{actual} \times K_{ideal}$$

$$= \text{number of scheduled resource units / number of available resource units}$$

$$= \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{r=1}^{[R]} Q_{jrt}/(T[R])$$

$0 <= \mathit{Efficiency} <= 1$

For example, the set of jobs to be scheduled by a mechanism require 80% of the available resources. It succeeds in scheduling 90% of its required resources. The *Efficiency* of the mechanisms is:

.9 x .8 = .72

2.1 Job-Scheduling Examples

We will illustrate this notion of efficiency with two simplified network job-scheduling mechanisms. The first mechanism (mechanism #1) knows how to utilize multiple CPUs. The second mechanism only knows how to schedule jobs sequentially on one CPU. There are two data sets to be measured against each mechanism. Since mechanism 1 is smarter, we expect it to be more efficient than mechanism 2.

Table 2: Data Set #1

	CPU	% of Memory	% of Total Bandwidth	Deadline (time units)	Resource Units Required
Job 1	8 units	50 (4 units)	50 (4 units)	10	16
Job 2	5 units	50 (2.5 units)	50 (2.5 units)	8	10
Total	13 units	6.5 units	6.5 units	--	26

Data set one has two jobs. The first job requires 8 time units of CPU usage (this could be qualified for different CPU speeds), meaning that the job can finish in 8 time units if it has full access to a CPU. This equates to 8 resource units. It requires 50% of the available memory while it is executing, equating to 4 resource units of memory. It requires 50% of the network bandwidth while it is executing, again equating to 4 resource units of bandwidth. Job 1 requires completion in 10 time units after starting. Job two is similar, except that it requires less CPU time, and has a shorter deadline. The two jobs require a total of 26 resource units. The length of the longest deadline (T) is 10. For resources, there are two CPUs, memory and network bandwidth, each 100% available for the duration of T, yielding 40 available resource units[1].

The second data set is the same as the first, except that the deadline for job 2 is increased to 14 time units. There are 52 available resource units.

Table 3: Data Set #2

	CPU	% of Memory	% of Total Bandwidth	Deadline	Resource Units Required
Job 1	8 units	50% (4 units)	50% (4 units)	10	16
Job 2	5 units	50% (2.5 units)	50% (2.5 units)	14	10
Total	13 units	6.5 units	6.5 units	--	26

Table 4 shows the result of submitting data set 1 to mechanism 1. The efficiency of this mechanism/data set is:

Efficiency = 26/40 = .65

Table 4: Mechanism 1, Data Set 1, T = 10, 40 available units

	1	2	3	4	5	6	7	8	9	10
CPU 1	■	■	■	■	■	■	■	■		
	■	■	■	■	■	■	■	■		

[1] To reflect realistic conditions, some or all of the available resources may be estimated to be less than 100%.

Table 4: Mechanism 1, Data Set 1, T = 10, 40 available units

CPU 2	▨	▨	▨	▨	▨					
	▨	▨	▨	▨	▨					
Memory	■	■	■	■	■	■	■	■		
	▨	▨	▨	▨	▨					
Bandwidth	■	■	■	■	■	■	■	■		
	▨	▨	▨	▨	▨					

Job 1	■
Job 2	▨

Table 5 shows the result of submitting data set 1 to mechanism 2. Here we see that the mechanism was not smart enough to utilize the second CPU, so Job 2 did not get scheduled at all. The efficiency of this mechanism is:

Table 5: Mechanism 2, Data Set 1, T = 10, 40 available units

	1	2	3	4	5	6	7	8	9	10
CPU 1	■	■	■	■	■	■	■	■		
	■	■	■	■	■	■	■	■		
CPU 2										
Memory	■	■	■	■	■	■	■	■		
Bandwidth	■	■	■	■	■	■	■	■		

$Efficiency = 16/40 = .40$

Table 6 and Table 7 show the result of submitting data set 2 to both mechanisms. Both jobs have enough time to finish the attempted jobs, so they both have the same efficiency. However, notice that in Table 6 , Mechanism 1 is not as efficient as it is in Table 4 , because the data set in Table 6 is easier.

$Efficiency = 26/52 = .5$

Table 6: Mechanism 1, Data Set 2, T = 13, 52 available units

	1	2	3	4	5	6	7	8	9	10	11	12	13
CPU 1	■	■	■	■	■	■	■	■					
	■	■	■	■	■	■	■	■					
CPU 2	▨	▨	▨	▨	▨								
	▨	▨	▨	▨	▨								
Memory	■	■	■	■	■	■	■	■					
	▨	▨	▨	▨	▨								

Table 6: Mechanism 1, Data Set 2, T = 13, 52 available units

Bandwidth	■	■	■	■	■	■	■	■					
	▨	▨	▨	▨	▨								

Table 7: Mechanism 2, Data Set 2, T = 13, 52 available units

	1	2	3	4	5	6	7	8	9	10	11	12	13
CPU 1	■	■	■	■	■	■	■	■	▨	▨	▨	▨	▨
	■	■	■	■	■	■	■	■	▨	▨	▨	▨	▨
CPU 2													
Memory	■	■	■	■	■	■	■	■	▨	▨	▨	▨	▨
Bandwidth	■	■	■	■	■	■	■	■	▨	▨	▨	▨	▨

3 A Network Scheduler Benefit Function

In this section, we will pursue the notion of a network benefit function which reflects resource and priority policies and utilizes the efficiency framework already established. This serves as an example of our approach; other resource usage policies could also be characterized.

A simple objective function that measures how well a network resource scheduler performs from a QoS point of view can be expressed as follows [1] [5], where B is an abstract per-job "user and system benefit function:"

$$max\sum_{j} B_j$$

That is to say, the network scheduler will be judged as to how well it meets the goals of the system and the users, as reflected in an expression of B.

3.1 Priority and Preference

Some jobs can produce output in different formats[1], where a given format (e.g., high resolution video) might be more resource consumptive than another format (e.g., low resolution video). A Quality of Service (QoS) scheduling mechanism might choose one format for a job over another, depending on varying network conditions (e.g., traffic congestion). The set of formats is represented by F. Different output formats may have different *preferences* (e.g., assigned by a user or "hard wired" as part of the application or job-scheduler database), and different levels of resource usage.

[1] For the sake of simplicity, factors relating to "format" have not been included in the examples of the benefit function in Section 2. The inclusion of variable output formats for jobs results in an additional dimension of resource usage and efficiency. Development of this dimension in the expressions of K_{ideal}, K_{actual}, $E_{fficiency}$, and Table 1 , is left to the reader.

The formats for a job are assigned preferences (p) by the user

where $0 \le p \le 1$ and
m_j is the number of <format, preference> pairs assigned for job j
p_{fj} is the preference the user has assigned to format f, job j

the preferences for a job (j) add up to 1: $\forall j \in J \sum_{f=1}^{m_j} p_{fj} = 1$

Jobs are assigned priorities for use in resolving resource contention and allocation issues. For example, a critical production job might be assigned a higher priority than an optional job. Priorities are typically administratively assigned. In other words, priorities are used to order jobs, whereas preferences are used to order formats for a particular job.

P_j is the priority of job j, where $0 \le P_j \le 1$

A network job scheduler should receive more credit in the benefit function for scheduling high priority and high preference jobs, as opposed to low priority or low preference jobs. We claim that a scheduler is intuitively doing a better job if important jobs, as judged by priority and preference, receive more attention (viz, resources) than unimportant jobs. How much weight the priorities and preferences are given is a matter of network scheduling policy.

3.2 Benefit Function

To begin with, the expression of *Efficiency* (see page 3) is simplified with a substitution:

let the expression of efficiency for a particular job (j) and format (f) be:

$$E_{fj} = \sum_{t=1}^{T} \sum_{r=1}^{[R]} Q_{fjrt} / ([R]T)$$

then the expression of efficiency for a scheduler over all formats and jobs becomes:

$$Efficiency = \sum_{f=1}^{mj} \sum_{j=1}^{n} E_{fj}$$

We represent preference (p) and priority (P) in a benefit function by averaging them in with the expression of efficiency (E) as follows[1]:

[1] Similar notions of priority, format and preference in measuring network efficiency have been proposed [6] for the MSHN project, however, "*B*" defines different relationships among these elements.

$$B= \frac{\sum_{j=1}^{n} \sum_{f=1}^{m_j} X_{fj}(P_j + p_{fj} + E_{fj})}{3n}$$

Where the characteristic function X is defined for f, j as:

$X_{fj} = 1$ if format f was successfully delivered to job j within time T_j, else 0

subject to: $\forall j \in J\left(\sum_{f=1}^{m_j} X_{fj} \leq 1\right)$

--at most one format is completed per job; f represents a particular <format, preference> pair[1]

3.3 Network Usage Policies

Network usage policies can vary widely. For example, restrictive usage policies (which include economic policies) attempt to moderate resource usage with usage-cost factors. In contrast, a priority-based policy emphasizes priority, diminishing the importance of a job's resource consumption. An ISP might manage its network of customers with an economic policy. A military information center might utilize a priority policy. ISP customers would be motivated not to "hog" network resources, by the cost incurred; whereas the military might want to ensure bandwidth to critical command-related jobs, at any cost.

Consider the analogy of water usage. The network is like a river, with citizens on the banks consuming water. The ISP and the military, in our example, are separate communities, and each have their own rivers. A set amount of water (viz., the resource) is flowing by and is available for use. The water that is not used flows past the community and is gone forever (here, our analogy breaks, as rivers are actually part of a globally replenishing cycle). So there is not a motivation to "conserve" water, other than to ensure that there is enough to go around at any given moment, and that it is allotted fairly. What is "fair" in a given community constitutes their water usage policy. Now, each community might have enough citizens to consume all of the water at periods of great usage. The ISP moderates the use of water, establishing a graduated policy: the first n units of water are charged at a low rate per gallon, while consumption above this limit is charged at a higher rate. The military (in this example) allows unlimited use to those with the highest priority, and divides up what is left among the lower-priority users.

If there is a drought, the river runs low. This is analogous to periods when there are equipment failures on a network.

3.4 Considering Resource Usage in the Benefit Function

In order to reflect a restrictive usage policy, we will modify the benefit function to give more credit to the scheduler for minimizing resource consumption. In other words, not only will the

[1] This expression reflects the simplifying assumption that a job has the same <format, preference> pair throughout its execution; whereas an adaptive RMS might enable changing format during the execution of the job.

scheduler score high for maximizing priority, preference and scheduling goals, it will also do better if it meets these goals using fewer resources. This resource usage policy will facilitate the addition of new jobs to a set of running jobs, to the extent that it motivates the availability of resources (Venkatasubramanian and Nahrstedt consider both resource consumption and user satisfaction in developing their metric of video QoS [8]).

A user may give a high preference to a job that requires high resource usage. We modify the preference term (*p*) with a representation of the job's (*j*) required resource usage[1], for a given format (*f*).

let $R_{fj} = 1 - K_{ideal.f.j}$

$p^*_{fj} = (p_{fj} + R_{fj}) / 2$

where $0 \leq R_{fj} \leq 1$, and a low R_{fj} indicates high resource usage. Note that in our formulation we have given equal weight to p_{fj} and r_{fj}; other weightings are possible

Table 8 shows an example of this modification to p. Here, we see preferences modified according to resource usage. There is no change for preferences whose values equal their resource usage values. A high preference job (.9) with high resource usage (.1) has a reduced preference; whereas, a low preference job (.1) with low usage (.9) results in an increased preference.

Table 8: Preference Modified by Resource Usage

		Job Preference			
		.1	.3	.6	.9
Resource Usage	.1	.1	.2	.35	.5
	.3	.2	.3	.45	.6
	.6	.35	.45	.6	.75
	.9	.5	.6	.75	.9

Using this modified preference value, the scheduler will receive more credit for scheduling jobs that combine both high user preferences and low resource usages.

3.5 Final Expression of Benefit Function

Different priority policies can also be represented by giving more or less weight to the job's priority, with a policy weighting factor (integer W), $W \geq 0$:

$$B = \frac{\sum_{j=1}^{n} \sum_{f=1}^{m_j} X_{fj}(WP_j + p^*_{fj} + E_{fj})}{n(W+2)}$$

[1] K_{ideal}, from Section 2, is expanded here with respect to job and format

$$0 \leq B \leq 1$$

The benefit function now reflects the average of *Efficiency*, preference and priority of the jobs that are submitted, as modified by the policies for priority and resource usage.

4 Summary

An approach for characterizing resource usage has been presented. This approach was used to develop a metric for resource-usage efficiency. The metric is applicable in the context of our ongoing work to represent security in an RMS benefit function [6], and to articulate a costing framework for security, that, for example, might be provided to a resource management system [4]. We have illustrated an example of applying this metric to two simplified schedulers. The efficiency metric was then combined with expressions of priority and preference to create a benefit function (B) which would reflect scheduler effectiveness in meeting user and system goals. Resource usage was also used to modify the user preference variable to indicate an economic resource management model.

Symbols Summary

B = benefit function

C_{jrt} = amount of resource r required by job j in time t

E_{fj} = efficiency of a particular job and format

Efficiency = efficiency of a network job-scheduling mechanism

F = {formats}

J = {jobs}

K_{ideal} = fraction of total resources required by a job data set

K_{actual} = fraction of required resources scheduled for a job data set

m_j = number of <format, preference> pairs for job j

P_j = priority of job j

p_{fj} = user preference for format f in job j

p^*_{fj} = user preference modified with respect to R

Q_{jrt} = amount of resource r consumed by job j in time t

R_{fj} = inverse of K_{ideal} for a given job/format

R = set of schedulable resources

[R] = number of distinct resources

T = latest deadline in a job data set

T_j = deadline for job j

W = policy weighting factor for priorities

X_{fj} = indicates if format f was successfully delivered to job j within time T_j

References

[1] Chatterjee, S., Sabata, B., Sydir, J. "ERDoS QOS Architecture," SRI Technical Report, ITAD-1667-TR-98-075, DARPA/ITO Contract Number N66001-97-C-8525, May 1998.

[2] Debra Hensgen, Taylor Kidd, David St. John, Matthew C. Schnaidt, H. J. Siegel, Tracy Braun, Jong-Kook Kim, Shoukat Ali, Cynthia Irvine, Tim Levin, Viktor Prasanna, Prashanth Bhat, Richard Freund, and Mike Gherrity, An Overview of the Management System for Heterogeneous Networks (MSHN), 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, Apr.1999

[3] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[4] Irvine, C., and Levin, T., Toward a Taxonomy and Costing Method for Security Metrics, NPS Technical Report, Forthcoming

[5] E. Jensen, C. Locke, and H. Tokuda, "A Time-driven Scheduling Model for Real-Time Operating Systems," in Proceedings of the IEEE Real-Time Systems Symposium, December 1985.

[6] Kim, Jong-Kook, Hensgen, D., Kidd, T., Siegel, H.J., St.John, D., Irvine, C., Levin, T., Prasanna, V., and Freund, R., Priorities, Deadlines, Versions, and Security in a Performance Measure Framework for Distributed Heterogeneous Networks, 8th IEEE International Symposium on High Performance Distributed Computing, Aug. 1999, Redondo Beach, Ca.

[7] Levin, T., and Irvine C., Quality of Security Service in a Resource Management System Benefit Function, NPS Technical Report, Forthcoming

[8] Vendatasubramanian, N. and Nahrstedt, K., "An Integrated Metric for Video QoS." ACM International Multimedia Conference, Seattle, Wa., Nov. 1997.

[9] Wright, R., Integrity Architecture and Security Services Demonstration for Management System for Heterogeneous Networks, Masters Thesis, Naval Postgraduate School, Monterey, CA, Sept. 1998.

[10] Wright, R., Shifflett, D., and Irvine, C. E., "Security Architecture for a Virtual Heterogeneous Machine," Proc. Computer Security Applications Conference, Dec. 1998.

Quality of Security Service in a Resource Management System Benefit Function

Tim Levin
Anteon Corporation
Monterey, CA 93940 USA

Cynthia Irvine
Naval Postgraduate School
Monterey, CA 93943 USA

Abstract: *Enforcement of a high-level statement of security policy may be difficult to discern when mapped through functional requirements to a myriad of possible security services and mechanisms in a highly complex, networked environment. A method of articulating network security functional requirements, and their fulfillment, is presented. Using this method, security in a quality of service framework is discussed in terms of "variant" security mechanisms and dynamic security policies. For illustration, it is shown how this method can be used to represent Quality of Security Service (QoSS) in a network scheduler benefit function*[1].

1 Introduction

Several efforts are underway to develop middleware systems that will logically combine network resources to construct a "virtual" computational system [4] [7] [8] . These geographically distributed, heterogeneous resources are expected to be used to support a heterogeneous mix of applications. Collections of tasks with disparate computation requirements will need to be efficiently scheduled for remote execution. Large parallelized computations found in fields such as astrophysics [14] and meteorology will require allocation of perhaps hundreds of individual processes to underlying systems. Multimedia applications, such as voice and video will impose requirements for low jitter, minimal packet losses, and isochronal data rates. Adaptive applications will need information about their environment so they can adjust to changing conditions.

User acceptance of these virtual systems, for either commercial or military applications, will depend, in part, upon the security, adaptability, and user-responsiveness provided. Several of the projects engaged in building the middleware to create these networks are pursuing the integration of security [6] [10] [22] and quality of service [1] [16] into these systems. The need for virtual networked systems to both adapt to varying security conditions, and offer the user a range of security choices is apparent.

In the network computing context, users or user programs may request the execution of "jobs," which are scheduled by an underlying control program to execute on local or remote computing resources. The execution of the job may access or consume a variety of network resources, like: local I/O device bandwidth, internetwork bandwidth; local and remote CPU time; local, intermediate (e.g., routing buffers) and remote storage. The resource usages may be temporary or persis-

[1] This work was funded through the DARPA/ITO Quorum program.

tent. As there are multiple users accessing the same resources, there are naturally various allotment, contention, and security issues regarding use of those resources.

The body of rules for resolving network security issues is called the network security policy, whereas the body of rules for resolving network contention and allotment comprise a network management policy (which is sometimes taken to include the network security policy). These policies consist of broad policy jurisdictions, such as scheduling, routing, access control, auditing, and authentication. Furthermore, these jurisdictions can be decomposed, typically, into functional requirements, such as, "users from network domain A must not access site B," and "user C must receive a certain quality of service." The network management and security policies, as mapped through the functional requirements, may be manifested in mechanisms throughout the network, including: host computer operating systems, network managers, traffic shapers, schedulers, routers, switches and combinations thereof. As these mechanisms are distributed and are often obscurely related, there has been some interest in the ability to express and quantify the level of support for security policy and Quality of Security Service (QoSS: managing security and security requests as a responsive "service" for which quantitative measurement of service "efficiency" is possible) provided in networked systems.

The purpose of this paper is to present a system for describing functional requirements for network security policies, to show how QoSS parameters and mechanisms can be represented in such a system, and to provide an example of the use of this system. The remainder of this paper is organized as follows. Section 2 discusses a "security vector" for quantifying functional support of network security policy. Section 3 describes how the security vector can be used for expressing the effects of QoSS in a network-scheduling benefit function; and a conclusion follows in Section 4.

2 Network Security Vector

A *network security policy* can be viewed as an n-dimensional space of functional security requirements. We represent this multidimensional space with a *vector* (S) of security components. Each component (S.c) specifies a boolean functional requirement, whereby the instantiation of a network job either meets (possibly trivially) or does not meet each of the requirements. By convention, a security vector's components are ordered, so they can be referenced ordinally (S.3) or symbolically (S.c). A component may indicate positive requirements (e.g., communications via node n must use encryption) as well as negative constraints (e.g., users from subnet s may not use node n). Components can also be hierarchically grouped [21] . Requirements for a given security service may be represented by one or more components (indicating a service *sub-vector*), and a security service may utilize functions and requirements of other services and their components.

Some jobs can produce output in different *formats*, where a given format (e.g., high resolution video) might be more resource consumptive than another format (e.g., low resolution video). Formats may have differing security requirements, even within the same job. For example, a video-stream format may require less packet authentication [18] , percentage-wise, than a series of fixed images based on the same data. A "quality of service" scheduling mechanism might choose one format for a job over another, depending on varying network conditions (e.g., traffic congestion). Further, adaptive applications may select formats depending upon changing conditions. For IPSec, security association (SA) processing using ISAKMP under IKE can permit complex security choices through an SA payload. For example, the payload recipient may be given transform choices regarding both Authentication Header and Encapsulating Security Protocol [13] .

The set of all jobs is represented by *J*. The set of all formats is represented by *I*. The notation S_{ij} identifies a vector containing the portions of *S* that are applicable to job *j* in format *i*, and $S_{ij}.c$ identifies a given component (*c*) of S_{ij}. The relation of *S* to S_{ij} is clarified further, below. The following are some informal examples of security-vector components:

- S.1: user access to resource r = RW; based on table t

- S.2: % of packets authenticated >= 50, <= 90; inc 10
- S.3: level (user) = level(resource)
- S.4: length of confidentiality encryption key >= 64, <= 256; inc 64
- S.5: authentication header transform *in* {HMAC-MD5, HMAC-SHA}

Here, "inc 10" indicates that the range from 50 through 90 is quantized into increments of 10, viz: 50, 60, 70, 80, 90. Later, we will need to indicate the number of quantized steps in the component; to do this, one more notational element is introduced, [*S.c*]. In the above examples, [*S.1*] = 1, and [*S.2*] = 5.

2.1 Variant Security Components

When [S.c] > 1, the underlying control program has a range within which it may allow the job to execute with respect to the policy requirement. We refer to this type of policy, and component, as "variant." Variant policies may be used within a resource management context, for example, to effect adaption to varying network conditions [17] . Also, if the policy mechanism is variant, the control program may offer QoSS choices to the user to indicate their preferences with respect to a given job or jobs. Without variant mechanisms, neither security adaptability by the underlying control program nor QoSS are possible, since fixed policy mechanisms do not allow for changes to security within a fixed job/resource environment. While the expression S.c may contain a compound boolean statement (see Section 2.2), by convention it may contain only one variant clause.

2.2 Component Structure

For use in the examples in this discussion, a component has the following composition (see Table 1 for details)[1]:

- component ::= boolean expression, variant-range-specifier ; modifying-clause
- boolean _expression ::= boolean_statement [(or | and) boolean_statement]*
- boolean_statement ::= LHS boolean-operator RHS

Table 1: Simple Component Elements

Element Name	Example S.1	Example S.2
Value	user access to resource r = RW, based on table t	% of packets authenticated >= 50, <= 90; inc 10
Instantiated value	false	true
Value of LHS	user access to resource r	% of packets authenticated
Instantiated value of LHS	W	70
Boolean operator	=	>=
Value of RHS	RW	50
variant range specifier		<= 90
Modifying clause	based on table t	inc 10

[1] It is not the focus here to elaborate on a policy representation language. See other efforts and works in progress [2] [3] [5] [15] .

A given policy component has a *value* which is a boolean *expression.* This component may also have an *instantiated value* with respect to a specific job and format, which is either "true" or "false." A component has a left hand side (LHS), which is the item that is being tested; of course the LHS has a *value* as well as an *instantiated value*. A component also has a right hand side (RHS), which is what the LHS is tested against, as well as zero or more modifying clauses. Similarly to the LHS, the RHS may have a value (or values) which is dependent on the instantiation of the component.

2.3 Dynamic Security Policies

With a dynamic security policy, the value of a vector's components may depend on the network "mode" (e.g., normal, impacted, emergency, etc.), where M is the set of all modes. There is, conceptually, a separate vector for each operational mode, represented as: S^{mode}. Access to a predefined set of alternate security policies allows their functional requirements and implementation mechanisms to be examined with respect to the overall policy prior to being fielded, rather than depending on ad hoc methods, for example, during an emergency.

Initially, every component of S has the same value in each of its modes. Ultimately, components may be assigned different values, depending on the network mode. For example:

- S^{normal}.b: user access to network node = granted; based on table t
- $S^{impacted}$.b: user access to network node = granted; based on table t, OR UID in set of administrators
- $S^{emergency}$.b: UID in set of {administrators, policymakers}

If a mode is not specified for a component (e.g., "S.a"), normal mode is assumed. This will be the case (i.e., the mode is unspecified) for the remainder of this discussion.

2.4 Refinements to S

R is the set of resources $\{r_1.. r_n\}$. R_{ij} is the subset of R utilized in executing job j in format i.

T_j is the requested completion time of job j.

Security policies may be expressed with respect to principals (user, group or role, etc.,), applications, data sets (both destination and source), formats, etc., as well as resources in R_{ij}.

The definition of S_{ij} is finally refined as follows: S_{ij} is a vector that is an order-preserving projection of S, such that a component c from S is in S_{ij} if and only if the value of c depends on format i, job j, or any r in R_{ij}. The number of components in a security vector S_{ij} *is* $[S_{ij}]$.

2.5 Summary of Security Vector

S is a general purpose notational system suitable for expressing arbitrarily complex sets of network security mechanisms. S can express variant policies, to allow security expressions of quality of service requests, and can have dynamic security elements to accommodate multiple situation-based policies. In particular, S can represent both (1) static security requirements that may be implemented in a system, as well as (2) the results of running a particular job or task against such static requirements. The latter usage will be examined in the next section, to express QoSS in an RMS benefit function.

3 Network-Scheduler Benefit Function

As discussed above, various mechanisms exist for managing contention for, and allotment of distributed network resources. One class of these mechanisms attempts to efficiently schedule the execution of multiple (possibly simultaneous) jobs on multiple distributed computers (e.g., the MSHN project [8] [22] [23] [11] [16]), where each job requires a determinable subset of the resources. Of interest is a benefit function for comparing the effectiveness of such job scheduling mechanisms when they are presented with real or hypothetical "data sets" of jobs.

Jobs are assigned *priorities* for use in resolving resource contention and allocation issues. In some systems, a job's priority may depend upon the particular operating *mode* of the network [8] . Also, the different data formats of a multiple-format job may have different *preferences* (e.g., assigned by a user or "hard wired" as part of the application or job-scheduler database), and different levels of resource usage [10] [12] . A network job scheduler should receive more credit in the benefit function for scheduling high priority and high preference jobs, as opposed to low priority or low preference jobs. That is to say, a scheduler is intuitively doing a better job if important jobs, as judged by priority and preference, receive more attention than unimportant jobs. How much weight the priorities and preferences are given is a matter of network scheduling policy.

We introduce a simple benefit function, *B*, to measure how well a scheduler meets the goals of user preference and system priorities (see [4] , [12] and [20] for other approaches). This function averages preference (*p*) and priority (*P*) (use of a priority and preference in measuring network effectiveness have been introduced for the MSHN project [10]) .

$$B= \frac{\sum_{j=1}^{n} \sum_{i=1}^{m_j} X_{ij}(P_{ij} + p_{ij})}{2n}$$

Where the characteristic function *X* is defined for *i, j* as:

Xij = 1 if format i was successfully delivered to job *j* within time T_j, else 0

and at most one format is completed per job: $\forall j \in J \left(\sum_{i=1}^{m_j} X_{ij} \leq 1 \right)$

Jobs and formats are defined as above, and P_j is the priority of job *j*, and $0 \leq P_j \leq 1$,

The formats for a job are assigned preferences (*p*) by the user, 0 <= *p* <= 1:

m_j is the number of {format, preference} pairs assigned for job *j*

p_{ij} is the preference the user has assigned to format *i*, job *j*

the preferences for a job add up to 1: $\forall j \in J: \sum_{i=1}^{m_j} p_{ij} = 1$

This approach assumes that users will assign preference values that correspond to resource usage, since we want the benefit function to indicate a higher value when the scheduler succeeds in scheduling "harder" jobs [12] .

3.1 Adding Security to the Benefit Function

We wish the benefit function to reflect the effectiveness and restrictions of the security policy. First, we define the characteristic security function *Z*, for *i* and *j*:

Z_{ij} = 1 if the instantiated value of all components in S_{ij} are true, else 0

The numerator of the benefit function is multiplied by *Z*, so that no credit is given for jobs that fail to meet the security requirements:

$$B= \frac{\sum_{j=1}^{n} \sum_{i=1}^{m_j} X_{ij} Z_{ij}(P_j + p_{ij})}{2n}$$

Now, for variant components, we wish to be able to give less credit to the scheduler for fulfilling less "difficult" security requirements. One algorithm for expressing this is for each instantiated component (*c*) in S_{ij} to be assigned a security completion token (*g*) where $0 \leq g \leq 1$. g_c will indicate the completion token corresponding to component *S.c*. g_c is defined to be the "percentage" of [*S.c*] met or exceeded by the *instantiated value* of the component's LHS (notated as *S.c"*): $g_c = S.c" / [S.c]$

To illustrate the calculation of g_1, for component S.1:

S.1: % of packets authenticated >= 50, <= 90; inc 10

[*S.1*] = 5 (the number of quanta in *S.1*), *S.1*" = 3 (the job achieves the 3rd quantum (70))

$g_1 = 3/5 = 0.6$

For invariant components, $g = 1$ or $g = 0$. A token (g) whose value is 0 represents a job "failing" the component's security policy. Recall that Z will be 0 when the job/format fails to meet the requirement of any security component, meaning that the function returns no benefit value for that job/format. We introduce a function (A) which averages the tokens of a job:

$A_{ij} = (g_1 + g_2 + .. + g_n)/n$

where n = $[S_{ij}]$ -- the number of components in $S_{ij,}$ and $(0 \le A_{ij} \le 1)$

We weight the tokens (g) assigned to individual security components, whereby each g_n has a corresponding integer weight (w_n), w_c >= 0. So *Aij* becomes:

$A_{ij} = (g_1w_1 + g_2w_2 + .. + g_nw_n)/(w_1 + w_2 + .. + w_n)$

The component weighting factors allow the benefit function to give more weight to components that are "more important" than others, e.g., reflecting network management policies.

In the final expression of the network benefit function, A is added to the numerator, providing an average of security, priority and preference.

$$B = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m_j} X_{ij} Z_{ij} (P_j + p_{ij} + A_{ij})}{3n}$$

$0 \le B \le 1$, where 1 indicates the maximum scheduling effectiveness.

4 Discussion and Conclusion

A security vector has been presented for describing functional requirements of network security policies. It has been shown that this vector can be used for representing security with respect to both quality of service and a network scheduler benefit function.

We are involved in ongoing work to organize the vector into a "normal form" with sub-vectors or hierarchies corresponding to policy jurisdictions (such as: scheduling, routing, access control, auditing, and authentication) and to incorporate a costing methodology for security components, such as can be provided to a resource management system [9] . We are working to develop a means of adjusting the preference expression with a notion of the corresponding resource usage [12] . We are considering how to expand the security benefit function (A) to reflect user quality of security service choices within the range allowed by variant security components, and to reflect performance implications of redundant security mechanisms.

The organizational security policy [19] governing the network may allow individuals or principals representing them to override rules represented by invariant security vector components. For example, a military commander might decide to forgo cryptographic secrecy mechanisms for a job in an emergency (e.g., to improve network performance), even though the system has not been configured with "dynamic" or "variant" security mechanisms, as defined herein. From the perspective of the security vector S and the benefit function, this is a *change or violation of the computer security policy*, which is not represented within the notation. It is recommended that this type of policy change be audited.

References

[1] Aurrecoechea, C., Campbell, A., and Hauw, L. "A Survey of Quality of Service Architectures", Multimedia Systems Journal, Special Issue on QoS Architectures, 1996.

[2] Badger, L., Stern, D. F., Sherman, D. L., Walker, K. M., and Haghighat, S. A., "Practical Domain and Type Enforcement for Unix," Proceedings of 1995 IEEE Symposium on Security and Privacy, 1995, Oakland, Ca., pp. 66-77

[3] Blaze, M., Feigenbaum, J., and Lacy, J., "Decentralized Trust Management," in Proceedings of 1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, Ca., pp 164-173

[4] Chatterjee, S., Sabata, B., Sydir, J. "ERDoS QOS Architecture," SRI Technical Report, ITAD-1667-TR-98-075, Menlo Park, CA, May 1998.

[5] Condell, M., Lynn, C. and Zao, J. "Security Policy Specification Language," INTERNET-DRAFT, Network Working Group, July 1, 1999, ftp://ftp.ietf.org/internet-drafts/draft-ietf-ipsec-spsl-01.txt, Expires January, 2000

[6] Foster, I, N. T. Karonis, N. T., Kesselman, C., Tuecke, S. Managing Security in High-Performance Distributed Computing. Cluster Computing 1(1):95-107, 1998.

[7] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[8] Debra Hensgen, Taylor Kidd, David St. John, Matthew C. Schnaidt, H. J. Siegel, Tracy Braun, Jong-Kook Kim, Shoukat Ali, Cynthia Irvine, Tim Levin, Viktor Prasanna, Prashanth Bhat, Richard Freund, and Mike Gherrity, An Overview of the Management System for Heterogeneous Networks (MSHN), 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, Apr.1999

[9] Irvine, C., and Levin, T., Toward a Taxonomy and Costing Method for Security Metrics, Annual Computer Security Applications Conference, Phoenix, AZ, to appear Dec. 1999

[10] Kim, Jong-Kook, Hensgen, D., Kidd, T., Siegel, H.J., St.John, D., Irvine, C., Levin, T., Porter, N.W., Prasanna, V., and Freund, R., A QoS Performance Measure Framework for Distributed Heterogeneous Networks, to appear in Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing, Rhodos, Greece, January 2000.

[11] Lee, C. Kesselman, C., Stepanek, j., Lindell, R., Hwang, S., Scott Michel, B., Bannister, J., Foster, I., and Roy, A. The Quality of Service Component for the Globus Metacomputing System. Proc. 1998 International Workshop on Quality of Service, Napa California, pp. 140-142, May, 1998.

[12] Levin, T., and Irvine C., An Approach to Characterizing Resource Usage and User Preferences in Benefit Functions, NPS Technical Report, NPS-CS-99-005

[13] Maughan, D., Schertler, M., Schneider, M., and Turner, J. Internet Security Association and Key Management Protocol, RFC 2408, http://info.internet.isi.edu/in-notes/rfc/files/rfc2408.txt

[14] Ostriker, J., and Norman. M. L., Cosmology of the Early Universe Viewed Through the New

Infrastructure. C.A.C.M. 40(11):85-94.

[15] Ryutov, T. and Neuman, C. Access Control Framework for Distributed Applications. INTERNET-DRAFT, CAT Working Group, USC/Information Sciences Institute, draft-ietf-cat-acc-cntrl-frmw-00.txt, August 07, 1998, Expires February 1999, http://gost.isi.edu/info/gaa_api.html

[16] Sabata, B., Chatterjee, S., Davis, M., Sydir, J., Lawrence, T. "Taxonomy for QoS Specifications," Proceedings the Third International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97), February 5-7, 1997, Newport Beach, Ca., pages 100-107

[17] Schantz, R. E. "Quality of Service," to be published in "Encyclopedia of Distributed Computing," 1998.

[18] Schneck, P. A., and Schwan, K., "Dynamic Authentication for High-Performance Networked Applications," Georgia Institute of Technology College of Computing Technical Report, GIT-CC-98-08, 1998.

[19] Stern, D. F., On the Buzzword "Security Policy", Proceedings of 1991 IEEE Symposium on Security and Privacy, 1991, Oakland, Ca., pages 219-230.

[20] Vendatasubramanian, N. and Nahrstedt, K., "An Integrated Metric for Video QoS," ACM International Multimedia Conference, Seattle, Wa., Nov. 1997.

[21] Wang, C. and Wulf, W. A, "A Framework for Security Measurement." Proc. National Information Systems Security Conference, Baltimore, MD, pp. 522-533, Oct. 1997.

[22] Wright, R., Integrity Architecture and Security Services Demonstration for Management System for Heterogeneous Networks, Masters Thesis, Naval Postgraduate School, Monterey, CA, Sept. 1998.

[23] Wright, R., Shifflett, D., and Irvine, C. E., "Security Architecture for a Virtual Heterogeneous Machine." Proc. Computer Security Applications Conference, Scottsdale, AZ, Dec. 1998, pp 167-17

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library, Code 013 2
Naval Postgraduate School
Monterey, CA 93943-5100

3. Research Office, Code 09 1
Naval Postgraduate School
Monterey, CA 93943-5000

4. Defense Advanced Research Project Agency/ 2
Information Technology Organization
ATTN: Dr. Gary Koob
3701 North Fairfax Drive
Arlington, VA 22203-1714

5. Professor Cynthia E. Irvine 6
Code CS/IC
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118

To appear in the proceedings of the *8th Heterogeneous Computing Workshop (HCW '99)*, cosponsors: IEEE Computer Society and Office of Naval Research, San Juan, Puerto Rico, Apr. 1999.

Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems

Muthucumaru Maheswaran[†], Shoukat Ali[‡], Howard Jay Siegel[‡],
Debra Hensgen[◇], and Richard F. Freund[*]

[†]Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2, Canada
Email: maheswar@cs.umanitoba.ca

[‡] School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285 USA
Email: {alis,hj}@ecn.purdue.edu

[◇]Department of Computer Science
Naval Postgraduate School
Monterey, CA 93940 USA
Email: hensgen@cs.nps.navy.mil

[*]NOEMIX Inc.
1425 Russ Blvd., Ste. T-110
San Diego, CA 92101 USA
Email: rffreund@noemix.com

Abstract

Dynamic mapping (matching and scheduling) heuristics for a class of independent tasks using heterogeneous distributed computing systems are studied. Two types of mapping heuristics are considered: on-line and batch mode heuristics. Three new heuristics, one for batch and two for on-line, are introduced as part of this research. Simulation studies are performed to compare these heuristics with some existing ones. In total, five on-line heuristics and three batch heuristics are examined. The on-line heuristics consider, to varying degrees and in different ways, task affinity for different machines and machine ready times. The batch heuristics consider these factors, as well as aging of tasks waiting to execute. The simulation results reveal that the choice of mapping heuristic depends on parameters such as: (a) the structure of the heterogeneity among tasks and machines, (b) the optimization requirements, and (c) the arrival rate of the tasks.

1. Introduction

An emerging trend in computing is to use distributed heterogeneous computing (HC) systems constructed by networking various machines to execute a set of tasks [5, 14]. These HC systems have resource management systems (RMSs) to govern the execution of the tasks that arrive for service. This paper describes and compares eight heuristics that can be used in such an RMS for assigning tasks to machines.

In a general HC system, dynamic schemes are necessary to assign tasks to machines (matching), and to compute the execution order of the tasks assigned to each machine (scheduling) [3]. In the HC system considered here, the tasks are assumed to be independent, i.e., no communications between the tasks are needed. A dynamic scheme is needed because the arrival times of the tasks may be random and some machines in the suite may go off-line and new machines may come on-line. The dynamic mapping (matching and scheduling) heuristics investigated in this study are non-preemptive, and assume that the tasks have no deadlines or priorities associated with them.

The mapping heuristics can be grouped into two categories: on-line mode and batch-mode heuristics. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The independent set of tasks that is considered for mapping at the mapping events is called a meta-task. A meta-task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and the ones that were mapped in earlier mapping events but did not begin execution. While on-line mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

The trade-offs between on-line and batch mode heuristics are studied experimentally. Mapping independent tasks onto an HC suite is a well-known NP-complete problem if throughput is the optimization criterion [9]. For the heuristics discussed in this paper, maximization of the throughput is the primary objective. This performance metric is the most common one in the production oriented environments. However, the performance of the heuristics is examined using other metrics as well.

Three new heuristics, one for batch and two for on-line,

This research was supported by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-97-M-0900, N62271-98-M-0217, and N62271-98-M-0448. Some of the equipment used was donated by Intel.

are introduced as part of this research. Simulation studies are performed to compare these heuristics with some existing ones. In total, five on-line heuristics and three batch heuristics are examined. The on-line heuristics consider, to varying degrees and in different ways, task affinity for different machines and machine ready times. The batch heuristics consider these factors, as well as aging of tasks waiting to execute.

Section 2 describes some related work. In Section 3, the optimization criterion and another performance metric are defined. Section 4 discusses the mapping approaches studied here. The simulation procedure is given in Section 5. Section 6 presents the simulation results.

This research is part of a DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks) [8]. MSHN is a collaborative research effort that includes Naval Postgraduate School, NOEMIX, Purdue, and University of Southern California. It builds on SmartNet, an operational scheduling framework and system for managing resources in an HC environment developed at NRaD [6]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service. The heuristics developed here, or their derivatives, may be included in the Scheduling Advisor component of the MSHN prototype.

2. Related Work

In the literature, mapping tasks onto machines is often referred to as scheduling. Several researchers have worked on the dynamic mapping problem from areas including job shop scheduling and distributed computer systems (e.g., [10, 12, 18, 20]).

Some of the heuristics examined for batch-mode mapping in this paper are based on the static heuristics given in [9]. The heuristics presented in [9] are concerned with mapping independent tasks onto heterogeneous machines such that the completion time of the last finishing task is minimized. The problem is recognized as NP-complete and several heuristics are designed. Worst case performance bounds are obtained for the heuristics. The Min-min heuristic that is used here as a benchmark for batch mode mapping is based on the ideas presented in [9], and implemented in SmartNet [6].

In [10], a dynamic matching and scheduling scheme based on a distributed policy for mapping tasks onto HC systems is provided. A task can have several subtasks, and the subtasks can have data dependencies among them. In the scheme presented in [10], the subtasks in an application receive information about the subtasks in other applications only in terms of load estimates on the machines. Each application uses an algorithm that uses a weighting factor to determine the mapping for the subtasks. The weighting factor for a subtask is derived by considering the length of the critical path from the subtask to the end of the directed acyclic graph (DAG) that represents the application. If each application is an independent task with no subtasks, as is the case in this paper, then the scheme presented in [10] is not suitable, because the mapping criterion is designed to exploit information available in a DAG. Therefore, the scheme provided in [10] is not compared to the heuristics presented in this paper.

Two dynamic mapping approaches, one using a centralized policy and the other using a distributed policy, are developed in [12]. The centralized heuristic referred to therein as the global queue equalization algorithm is similar to the minimum completion time heuristic that is used as a benchmark in this paper and described in Section 4. The heuristic based on the distributed policy uses a method similar to the minimum completion time heuristic at each node. The mapper at a given node considers the local machine and the k highest communication bandwidth neighbors to map the tasks in the local queue. Therefore, the mapper based on the distributed strategy assigns a task to the best machine among the $k+1$ machines. The simulation results provided in [12] show that the centralized heuristic always performs better than the distributed heuristic. The heuristics in [12] are very similar to the minimum completion time heuristic used as a benchmark in this paper. Hence, they are not experimentally compared with the heuristics presented here.

In [18], a survey of dynamic scheduling heuristics for distributed computing systems is provided. Most of the heuristics featured in [18] perform load sharing to schedule the tasks on different machines, not considering any task-machine affinities while making the mapping decisions for HC systems. In contrast to [18], these affinities are considered to varying degrees in all but one of the heuristics examined in this paper.

A survey of dynamic scheduling heuristics for job-shop environments is provided in [20]. It classifies the dynamic scheduling algorithms into three approaches: conventional approach, knowledge-based approach, and distributed problem solving approach. The class of heuristics grouped under the conventional approach are similar to the minimum completion time heuristic considered in this paper, however, the problem domains considered in [20] and here differ. Furthermore, some of the heuristics featured in [20] use priorities and deadlines to determine the task scheduling order whereas priorities and deadlines are not considered here.

In distributed computer systems, load balancing schemes have been a popular strategy for mapping tasks onto the machines (e.g., [15, 18]). In [15], the performance characteristics of simple load balancing heuristics for HC distributed systems are studied. The heuristics presented in [15] do not consider task execution times when making their decisions.

SmartNet [6] is an RMS for HC systems that employs various heuristics to map tasks to machines considering resource and task heterogeneity. In this paper, some appropriate selected SmartNet heuristics are included in the comparative study.

3. Performance Metrics

The expected execution time e_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given m_j has no load when t_i is assigned. The expected completion time c_{ij} of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let m be the total number of the machines in the HC suite. Let K be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be a_i, and let the begin time of t_i be b_i. From the above definitions, $c_{ij} = b_i + e_{ij}$. Let c_i be c_{ij}, where machine j is assigned to execute task i. The makespan for the complete schedule is then defined as $max_{t_i \in K}(c_i)$ [17]. Makespan is a measure of the throughput of the HC system, and does not measure the quality of service imparted to an individual task.

Recall from Section 1, in on-line mode, the mapper assigns a task to a machine as soon as the task arrives at the mapper, and in batch mode a set of independent tasks that need to be mapped at a mapping event is called a meta-task. (In some systems, the term meta-task is defined in a way that allows inter-task dependencies.) In batch mode, for the i-th mapping event, the meta-task M_i is mapped at time τ_i, where $i \geq 0$. The initial meta-task, M_0, consists of all the tasks that arrived prior to time τ_0, i.e., $M_0 = \{t_j \mid a_j < \tau_0\}$. The meta-task, M_k, for $k > 0$, consists of tasks that arrived after the last mapping event and the tasks that had been mapped, but did not start executing, i.e., $M_k = \{t_j \mid \tau_{k-1} \leq a_j < \tau_k\} \cup \{t_j \mid a_j < \tau_{k-1}, b_j > \tau_k\}$. The waiting time for task t_j is defined as $b_j - a_j$. Let $\bar{c}_j$ be the completion time of task t_j if it is the only task that is executing on the system. The sharing penalty (ρ_j) for the task t_j is defined as $(c_j - \bar{c}_j)$. The average sharing penalty for the tasks in the set K is given by $[\sum_{t_j \in K} \rho_j]/\mid K \mid$. The average sharing penalty for a set of tasks mapped by a given heuristic is an indication of the heuristic's ability to minimize the effects of contention among different tasks in the set. It therefore indicates quality of service provided to an individual task, as gauged by the wait incurred by the task before it begins and the time to perform the actual computation. Other performance metrics are considered in [13].

4. Mapping Heuristics

4.1. Overview

In the on-line mode heuristics, each task is considered only once for matching and scheduling, i.e., the mapping is not changed once it is computed. When the arrival rate is low, machines may be ready to execute a task as soon as it arrives at the mapper. Therefore, it may be beneficial to use the mapper in the on-line mode so that a task need not wait until the next mapping event to begin its execution.

In batch mode, the mapper considers a meta-task for matching and scheduling at each mapping event. This enables the mapping heuristics to possibly make better decisions, because the heuristics have the resource requirement information for a whole meta-task, and know about the actual execution times of a larger number of tasks (as more tasks might complete while waiting for the mapping event). When the task arrival rate is high, there will be a sufficient number of tasks to keep the machines busy in between the mapping events, and while a mapping is being computed. It is, however, assumed in this study that the running time of the heuristic is negligibly small as compared to the average task execution time.

Both on-line and batch mode heuristics assume that estimates of expected task execution times on each machine in the HC suite are known. The assumption that these estimated expected times are known is commonly made when studying mapping heuristics for HC systems (e.g.,[7, 11, 19]). (Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [14].) These estimates can be supplied before a task is submitted for execution, or at the time it is submitted. (The use of some of the heuristics studied here in a static environment is discussed in [4].)

The ready time of a machine is quantified by the earliest time that machine is going to be ready after completing the execution of the tasks that are currently assigned to it. It is assumed that each time a task t_i completes on a machine m_j a report is sent to the mapper. Because the heuristics presented here are dynamic, the expected machine ready times are based on a combination of actual task execution times and estimated expected task execution times. The experiments presented in Section 6 model this situation using simulated actual values for the execution times of the tasks that have already finished their execution. Also, all heuristics examined here operate in a centralized fashion on a dedicated suite of machines; i.e., the mapper controls the execution of all jobs on all machines in the suite. It is also assumed that the mapping heuristic is being run on a separate machine.

4.2. On-line mode mapping heuristics

The MCT (minimum completion time) heuristic assigns each task to the machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The MCT heuristic is a variant of the fast-greedy heuristic from SmartNet [6]. The MCT heuristic is used as a benchmark for the on-line mode, i.e., the performance of the other heuristics is compared against that of the MCT heuristic.

As a task arrives, all the machines in the HC suite are examined to determine the machine that gives the earliest completion time for the task. Therefore, it takes $O(m)$ time to map a given task.

The MET (minimum execution time) heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time (this heuristic is also known as LBA (limited best assignment) [1] and UDA (user directed assignment) [6]). This heuristic, in contrast to MCT, does not consider machine ready times. This heuristic can cause a severe imbalance in load across the machines. The advantages of this method are that it gives each task to the machine that performs it in the least amount of execution time, and the heuristic is very simple. The heuristic needs $O(m)$ time to find the machine that has the minimum execution time for a task.

The SA (switching algorithm) heuristic is motivated by the following observations. The MET heuristic can potentially create load imbalance across machines by assigning many more tasks to some machines than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. If the tasks are arriving in a random mix, it is possible to use MET at the expense of load balance until a given threshold and then use MCT to smooth the load across the machines. The SA heuristic uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. The purpose is to have a heuristic with the desirable properties of both the MCT and the MET.

Let the maximum ready time over all machines in the suite be r_{max}, and the minimum ready time be r_{min}. Then, the load balance index across the machines is given by $\pi = r_{min}/r_{max}$. The parameter π can have any value in the interval $[0, 1]$. If π is 1.0, then the load is evenly balanced across the machines. If π is 0, then at least one machine has not yet been assigned a task. Two threshold values, π_l (low) and π_h (high), for the ratio π are chosen in $[0, 1]$ such that $\pi_l < \pi_h$. Initially, the value of π is set to 0.0. The SA heuristic begins mapping tasks using the MCT heuristic until the value of load balance index increases to at least π_h. After that point in time, the SA heuristic begins using the MET heuristic to perform task mapping. This causes the load balance index to decrease. When it reaches π_l, the SA heuristic switches back to using the MCT heuristic for mapping the tasks and the cycle continues.

As an example of functioning of the SA with lower and upper limits of 0.6 and 0.9, respectively, for $| K |= 1000$, the SA switched between the MET and the MCT two times, assigning 715 tasks using the MCT. For $| K |= 2000$, the SA switched five times, using the MCT to assign 1080 tasks. The percentage of tasks assigned using MCT gets progressively smaller for larger $| K |$. This is because an MET assignment in a highly loaded system will bring a smaller decrease in load balance index than when the same assignment is made in a lightly loaded system. Therefore many more MET assignments can be made in a highly loaded system before the load balance index falls below the lower threshold.

At each task's arrival, the SA heuristic determines the load balance index. In the worst case, this takes $O(m)$ time. In the next step, the time taken to assign a task to a machine is of order $O(m)$ whether SA uses the MET to perform the mapping or the MCT. Overall, the SA heuristic takes $O(m)$ time irrespective of which heuristic is actually used for mapping the task.

The KPB (k-percent best) heuristic considers only a subset of machines while mapping a task. The subset is formed by picking the $(km/100)$ best machines based on the execution times for the task, where $100/m \leq k \leq 100$. The task is assigned to a machine that provides the earliest completion time in the subset. If $k = 100$, then the KPB heuristic is reduced to the MCT heuristic. If $k = 100/m$, then the KPB heuristic is reduced to the MET heuristic. A "good" value of k maps a task to a machine only within a subset formed from computationally superior machines. The purpose is not as much as matching of the current task to a computationally well-matched machine as it is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks. This "foresight" about task heterogeneity lacks in the MCT which might assign a task to a poorly matched machine for an immediate marginal improvement in completion time, possibly depriving some subsequently arriving tasks of that machine, and eventually leading to a larger makespan as compared to the KPB. It should be noted that while both the KPB and SA have elements of the MCT and the MET in their operation, it is only in the KPB that *each* task assignment attempts to optimize objectives of the MCT and the MET simultaneously. However, in cases where a fixed subset of machines is not among the k% best for any task, the KPB will cause much machine idle time compared to the MCT, and can result in much poorer performance.

For each task, $O(m \log m)$ time is spent in ranking the machines for determining the subset of machines to examine. Once the subset of machines is determined, it takes

$O(\frac{km}{100})$ time, i.e., $O(m)$ time to determine the machine assignment. Overall the heuristic takes $O(m \log m)$ time.

The OLB (opportunistic load balancing) heuristic assigns the task to the machine that becomes ready next. It does not consider the execution time of the task when mapping it onto a machine. If multiple machines become ready at the same time, then one machine is arbitrarily chosen.

The complexity of the OLB heuristic is dependent on the implementation. In the implementation considered here, the mapper may need to examine all m machines to find the machine that becomes ready next. Therefore, it takes $O(m)$ to find the assignment. Other implementations may require idle machines to assign tasks to themselves by accessing a shared global queue of tasks [21].

4.3. Batch mode mapping heuristics

In the batch mode heuristics, meta-tasks are mapped after predefined intervals. These intervals are defined in this study using one of the two strategies proposed below.

The regular time interval strategy maps the meta-tasks at regular intervals of time except when all machines are busy. When all machines are busy, all scheduled mapping events that precede the one before the expected ready time of the machine that finishes earliest are canceled.

The fixed count strategy maps a meta-task M_i as soon as one of the following two mutually exclusive conditions are met: (a) an arriving task makes $|M_i|$ larger than or equal to a predetermined arbitrary number κ, or (b) all tasks have arrived, and a task completes while the number of tasks which yet have to begin is larger than or equal to κ. In this strategy, the length of the mapping intervals will depend on the arrival rate and the completion rate. The possibility of machines being idle while waiting for the next mapping event will depend on the arrival rate, completion rate, m, and κ.

The batch mode heuristics considered in this study are discussed in the paragraphs below. The complexity analysis performed for these heuristics considers a single mapping event. In the complexity analysis, the meta-task size is assumed to be equal to the average of meta-task sizes at all actually performed mapping events. Let the average meta-task size be S.

The Min-min heuristic shown in Figure 1 is from SmartNet [6]. In Figure 1, let r_j denote the expected time machine m_j will become ready to execute a task after finishing the execution of all tasks assigned to it at that point in time. First the c_{ij} entries are computed using the e_{ij} and r_j values. For each task t_i the machine that gives the earliest expected completion time is determined by scanning the rows of the c matrix. The task t_k that has the minimum earliest expected completion time is determined and then assigned to the corresponding machine. The matrix c and vector r are updated and the above process is repeated with tasks that have not yet been assigned a machine.

Min-min begins by scheduling the tasks that change the expected machine ready time status by the least amount that any assignment could. If tasks t_i and t_k are contending for a particular machine m_j, then Min-min assigns m_j to the task (say t_i) that will change the ready time of m_j less. This increases the probability that t_k will still have its earliest completion time on m_j, and shall be assigned to it. Because at $t = 0$, the machine which finishes a task earliest is also the one that executes it fastest, and from thereon Min-min heuristic changes machine ready time status by the least amount for every assignment, the percentage of tasks assigned their first choice (on basis of expected execution time) is likely to be higher in Min-min than with the other batch mode heuristics described in this section. The expectation is that a smaller makespan can be obtained if a larger number of tasks is assigned to the machines that not only complete them earliest but also execute them fastest.

```
(1) for all tasks $t_i$ in meta-task $M_v$ (in an arbitrary order)
(2)     for all machines $m_j$ (in a fixed arbitrary order)
(3)         $c_{ij} = e_{ij} + r_j$
(4) do until all tasks in $M_v$ are mapped
(5)     for each task in $M_v$ find the earliest completion
            time and the machine that obtains it
(6)     find the task $t_k$ with the minimum earliest
            completion time
(7)     assign task $t_k$ to the machine $m_l$ that gives the
(8)         earliest completion time
(9)     delete task $t_k$ from $M_v$
(10)    update $r_l$
(11)    update $c_{il}$ for all $i$
(12)enddo
```

Figure 1. The Min-min heuristic.

The initialization of the c matrix in Line (1) to Line (3) takes $O(Sm)$ time. The **do** loop of the Min-min heuristic is repeated S times and each iteration takes $O(Sm)$ time. Therefore, the heuristic takes $O(S^2m)$ time.

The Max-min heuristic is similar to the Min-min heuristic given in Figure 1. It is also from SmartNet [6]. Once the machine that provides the earliest completion time is found for every task, the task t_k that has the maximum earliest completion time is determined and then assigned to the corresponding machine. The matrix c and vector r are updated and the above process is repeated with tasks that have not yet been assigned a machine. The Max-min heuristic has the same complexity as the Min-min heuristic.

The Max-min is likely to do better than the Min-min heuristic in the cases where we have many more shorter tasks than the long tasks. For example, if there is only one long task, Max-min will execute many short tasks concurrently with the long task. The resulting makespan might just be determined by the execution time of the long task in these cases. Min-min, however, first finishes the shorter tasks (which may be more or less evenly distributed over the machines) and then executes the long task, increasing the makespan.

The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it. Let the sufferage value of a task t_i be the difference between its second earliest completion time (on some machine m_y) and its earliest completion time (on some machine m_x). That is, using m_x will result in the best completion time for t_i, and using m_y the second best.

Figure 2 shows the Sufferage heuristic. The initialization phase in Lines (1) to (3) is similar to the ones in the Min-min and Max-min heuristics. Initially all machines are marked unassigned. In each iteration of the **for** loop in Lines (6) to (14), pick arbitrarily a task t_k from the meta-task. Find the machine m_j that gives the earliest completion time for task t_k, and tentatively assign m_j to t_k if m_j is unassigned. Mark m_j as assigned, and remove t_k from meta-task. If, however, machine m_j has been previously assigned to a task t_i, choose from t_i and t_k the task that has the higher sufferage value, assign m_j to the chosen task, and remove the chosen task from the meta-task. The unchosen task will not be considered again for this execution of the **for** statement, but shall be considered for the next iteration of the **do** loop beginning on Line (4). When all the iterations of the **for** loop are completed (i.e., when one execution of the **for** statement is completed), update the machine ready time of the each machine assigned a new task. Perform the next iteration of the **do** loop beginning on Line (4) until all tasks have been mapped.

Table 1 shows a scenario in which the Sufferage will outperform the Min-min. Table 1 shows the expected execution time values for four tasks on four machines (all initially idle). In this particular case, the Min-min heuristic gives a makespan of 9.3 and the Sufferage heuristic gives a makespan of 7.8. Figure 3 gives a pictorial representation of the assignments made for the case in Table 1.

	m_0	m_1	m_2	m_3
t_0	4	4.8	13.4	5
t_1	5	8.2	8.8	8.9
t_2	5.5	6.8	9.4	9.3
t_3	5.2	6	7.8	10.8

Table 1. An example expected execution time matrix that illustrates the situation where the Sufferage heuristic outperforms the Min-min heuristic.

From the pseudo code given in Figure 2, it can be observed that first execution of the **for** statement on Line (6) takes $O(Sm)$ time. The number of task assignments made in one execution of this **for** statement depends on the total number of machines in the HC suite, the number of machines that are being contended for among different tasks, and the number of tasks in the meta-task being mapped. In the worst case, only one task assignment will be made in each execution of the **for** statement. Then meta-task size will decrease by one at each **for** statement execution. The outer **do** loop will be iterated S times to map the whole meta-task. Therefore, in the worst case, the time $T(S)$ taken to map a meta-task of size S will be

$$T(S) = Sm + (S-1)m + (S-2)m + \cdots + m$$

$$T(S) = O(S^2m)$$

In the best case, there are as many machines as there are tasks in the meta-task, and there is no contention among the tasks. Then all the task are assigned in the first execution of the **for** statement so that $T(S) = O(Sm)$. Let ω be a number quantifying the extent of contention among the tasks for the different machines. The running time of Sufferage heuristic can then be given as $O(\omega Sm)$ time, where $1 \leq \omega \leq S$. It can be seen that ω is equal to S in the worst case, and is 1 in the best case; these values of ω are numerically equal to the number of iterations of the **do** loop on Line (4).

```
(1)  for all tasks $t_k$ in meta-task $M_v$ (in an arbitrary order)
(2)      for all machines $m_j$ (in a fixed arbitrary order)
(3)          $c_{kj} = e_{kj} + r_j$
(4)  do until all tasks in $M_v$ are mapped
(5)      mark all machines as unassigned
(6)      for each task $t_k$ in $M_v$ (in an arbitrary order)
(7)          find machine $m_j$ that gives the earliest
                 completion time
(8)          sufferage value = second earliest completion
                 time − earliest completion time
(9)          if machine $m_j$ is unassigned
(10)             assign $t_k$ to machine $m_j$, delete $t_k$
                 from $M_v$, mark $m_j$ assigned
(11)         else
(12)             if sufferage value of task $t_i$ already
                 assigned to $m_j$ is less than the
                 sufferage value of task $t_k$
(13)                 unassign $t_i$, add $t_i$ back to $M_v$,
                     assign $t_k$ to machine $m_j$,
                     delete $t_k$ from $M_v$
(14)     endfor
(15)     update the vector $r$ based on the tasks that
             were assigned to the machines
(16)     update the $c$ matrix
(17) enddo
```

Figure 2. The Sufferage heuristic.

The batch mode heuristics can cause some tasks to be starved of machines. Let H_i be a subset of meta-task M_i consisting of tasks that were mapped (as part of M_i) at the mapping event i at time τ_i but did not begin execution by the next mapping event at τ_{i+1}. H_i is the subset of M_i that is included in M_{i+1}. Due to the expected heterogeneous nature of the tasks, the meta-task M_{i+1} may be so mapped that some or all of the tasks arriving between τ_i and τ_{i+1} may begin executing before the tasks in set H_i do. It is possible that some or all of the tasks in H_i may be included in H_{i+1}. This probability increases as the number of new tasks arriving between τ_i and τ_{i+1} increases. In general, some tasks may be remapped at each successive mapping event without actually beginning execution (i.e., the task is starving for a machine).

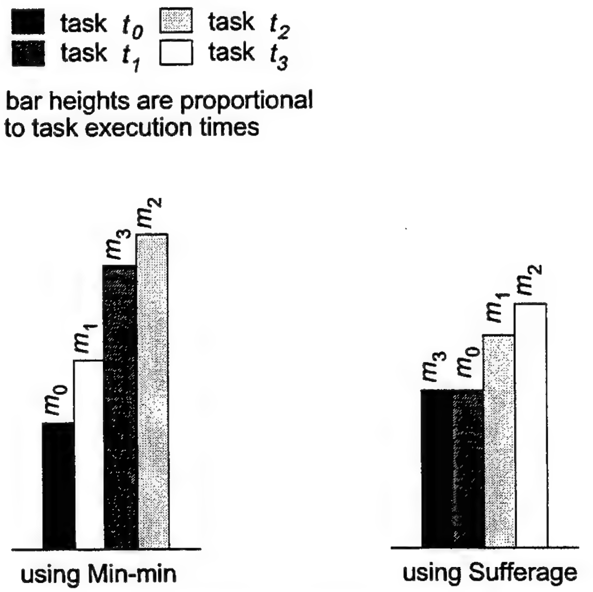


Figure 3. An example scenario (based on Table 1) where the Sufferage gives a shorter makespan than the Min-min.

To reduce starvation, aging schemes are implemented. The age of a task is set to zero when it is mapped for the first time and incremented by one each time the task is remapped. Let σ be a constant that can be adjusted empirically to change the extent to which aging affects the operation of the heuristic. An aging factor, $\zeta = (1 + \text{age}/\sigma)$, is then computed for each task. For the experiments in this study, σ is set to 10. The aging factor is used to enhance the probability of an "older" task beginning before the tasks that would otherwise begin first. In the Min-min heuristic, for each task, the completion time obtained in Line (5) of Figure 1 is multiplied by the corresponding value for $\frac{1}{\zeta}$. As the age of a task increases, its age-compensated expected completion time (i.e., one used to determine the mapping) gets increasingly smaller than its original expected completion time. This increases its probability of being selected in Line (6) in Figure 1.

Similarly, for the Max-min heuristic, the completion time of a task is multiplied by ζ. In the Sufferage heuristic, the sufferage value computed in Line (8) in Figure 2 is multiplied by ζ.

5. Simulation Procedure

The mappings are simulated using a discrete event simulator. The task arrivals are modeled by a Poisson random process. The simulator contains an ETC (expected time to compute) matrix that contains the expected execution times of a task on all machines, for all the tasks that can arrive for service. The ETC matrix entries used in the simulation studies represent the e_{ij} values that the heuristic would use in its operation. The actual execution time of a task can be different from the value given by the ETC matrix. This variation is modeled by generating a simulated actual execution time for each task by sampling a truncated Gaussian probability density function with variance equal to three times the expected execution time of the task and mean equal to the expected execution time of the task [2, 16]. If the sampling results in a negative value, the value is discarded and the same probability density function is sampled again. This process is repeated until a positive value is returned by the sampling process.

In an ETC matrix, the numbers along a row indicate the execution times of the corresponding task on different machines. The average variation along the rows is referred to as the machine heterogeneity [2]. Similarly, the average variation along the columns is referred to as the task heterogeneity [2]. One classification of heterogeneity is to divide it into high heterogeneity and low heterogeneity. Based on the above idea, four categories were proposed for the ETC matrix in [2]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo). The ETC matrix can be further classified into two classes, consistent and inconsistent, which are orthogonal to the previous classifications. For a consistent ETC matrix, if machine m_x has a lower execution time than machine m_y for task t_k, then the same is true for any task t_i. The ETC matrices that are not consistent are inconsistent ETC matrices. In addition to the consistent and inconsistent classes, a semi-consistent class could also be defined. A semi-consistent ETC matrix is characterized by a consistent sub-matrix. In the semi-consistent ETC matrices used here, 50% of the tasks and 25% of the machines define a consistent sub-matrix. Furthermore, it is assumed that for a particular task the execution times that fall within the consistent sub-matrix are smaller than those that fall out. This assumption is justified because the machines that perform consistently better than the others for some tasks are more likely to be very much faster for those tasks than very much slower.

Let an ETC matrix have $\underline{t_{max}}$ rows and $\underline{m_{max}}$ columns. Random ETC matrices that belong to the different categories are generated in the following manner:

1. Let $\underline{\Gamma_t}$ be an arbitrary constant quantifying task heterogeneity, being smaller for low task heterogeneity. Let $\underline{N_t}$ be a number picked from the uniform random distribution with range $[1,\Gamma_t]$.
2. Let $\underline{\Gamma_m}$ be an arbitrary constant quantifying machine heterogeneity, being smaller for low machine heterogeneity. Let $\underline{N_m}$ be a number picked from the uniform random distribution with range $[1,\Gamma_m]$.
3. Sample N_t t_{max} times to get a vector $\underline{q}[0..(t_{max}-1)]$.
4. Generate the ETC matrix, $\underline{e}[0..(t_{max}-1),0..(m_{max}-1)]$ by the following algorithm:

```
for $t_i$ from 0 to $(t_{max}-1)$
    for $m_j$ from 0 to $(m_{max}-1)$
        pick a new value for $N_m$
        $e[i,j] = q[i] * N_m$.
    endfor
endfor
```

From the raw ETC matrix generated above, a semi-consistent matrix could be generated by sorting the execution times for a random subset of the tasks on a random subset of machines. An inconsistent ETC matrix could be obtained simply by leaving the raw ETC matrix as such. Consistent ETC matrices were not considered in this study because they are least likely to arise in the current intended MSHN environment.

In the experiments described here, the values of Γ_t for low and high task heterogeneities are 1000 and 3000, respectively. The values of Γ_m for low and high machine heterogeneities are 10 and 100, respectively. These heterogeneous ranges are based on one type of expected environment for MSHN.

6. Experimental Results and Discussion

6.1. Overview

The experimental evaluation of the heuristics is performed in three parts. In the first part, the on-line mode heuristics are compared using various metrics. The second part involves a comparison of the batch mode heuristics. The third part is the comparison of the batch mode and the on-line mode heuristics. Unless stated otherwise, the following are valid for the experiments described here. The number of machines is held constant at 20, and the experiments are performed for $|K| = \{1000, 2000\}$. All heuristics are evaluated in a HiHi heterogeneity environment, both for the inconsistent and the semi-consistent cases, because these correspond to some of the currently expected MSHN environments. A Poisson distribution is used to generate the task arrivals. For each value of $|K|$, tasks are mapped under two different arrival rates, λ_h and λ_l, such that $\lambda_h > \lambda_l$. The value of λ_h is chosen empirically to be high enough to allow at most 50% tasks to complete when the last task in the set arrives. Similarly, λ_l is chosen to be low enough to allow at least 90% of the tasks to complete when the last task in the set arrives. The MCT heuristic is used in this standardization. Unless otherwise stated, the task arrival rate is set to λ_h. λ_l is more likely to represent an HC system where the task arrival is characterized by little burstiness; no particular group of tasks arrives in a much shorter span of time than some other group having same number of tasks. λ_h is supposed to characterize the arrivals in an HC system where a large group of tasks arrives in a much shorter time than some other group having same number of tasks; e.g., in this case a burst of $|K|$ tasks.

Example comparisons are discussed in Subsections 6.2 to 6.4. Each data point in the comparison charts is an average over 50 trials, where for each trial the simulated actual task execution times are chosen independently. More general conclusions about the heuristics' performance is in Section 7. Comparisons for a larger set of performance metrics are given in [13].

6.2. Comparisons of the on-line mode heuristics

Unless otherwise stated, the on-line mode heuristics are investigated under the following conditions. In the KPB heuristic, k is equal to 20%. This particular value of k was found to give the lowest makespan for the KPB heuristic under the conditions of the experiments. For the SA, the lower threshold and the upper threshold for the load balance index are 0.6 and 0.9, respectively. Once again these values were found to give optimum values of makespan for the SA.

In Figure 4, on-line mode heuristics are compared based on makespan for inconsistent HiHi heterogeneity. From Figure 4, it can be noted that the KPB heuristic completes the execution of the last finishing task earlier than the other heuristics (however, it is only slightly better than the MCT). For $k = 20\%$ and $m = 20$, the KPB heuristic forces a task to choose a machine from a subset of four machines. These four machines have the lowest execution times for the given task. The chosen machine would give the smallest completion time as compared to other machines in the set.

Figure 5 compares the on-line mode heuristics using average sharing penalty. Once again, the KPB heuristic performs best. However, the margin of improvement is smaller than that for the makespan. It is evident that the KPB provides maximum throughput (system oriented performance metric) and minimum average sharing penalty (application oriented performance metric).

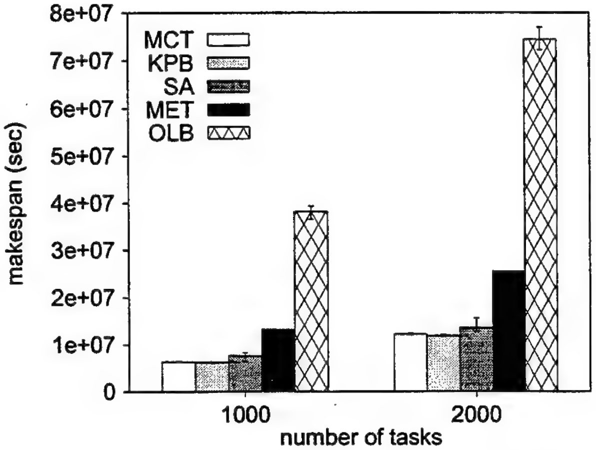


Figure 4. Makespan for the on-line heuristics for inconsistent HiHi heterogeneity.

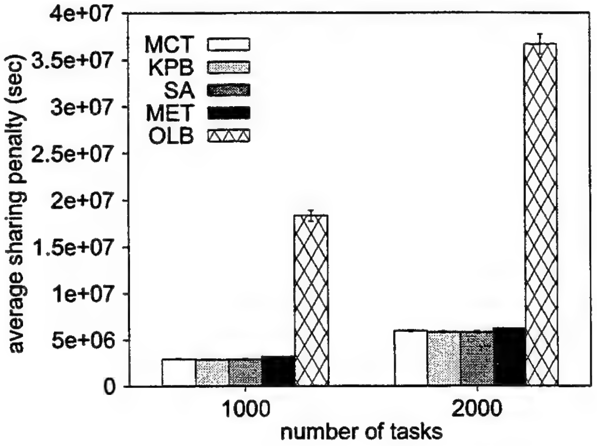


Figure 5. Average sharing penalty of the on-line heuristics for inconsistent HiHi heterogeneity.

Figure 6 compares the makespans of the different on-line heuristics for semi-consistent HiHi ETC matrices. Figure 7 compares the average sharing penalties of the different on-line heuristics. As shown in Figures 4 and 6 the relative performance of the different on-line heuristics is impacted by the degree of consistency of the ETC matrices.

For the semi-consistent type of heterogeneity, machines within a particular subset perform tasks that lie within a particular subset faster than other machines. From Figure 6, it can be observed that for semi-consistent ETC matrices, the MET heuristic performs the worst. For the semi-consistent matrices used in these simulations, the MET heuristic maps half of the tasks to the same machine, considerably increasing the load imbalance. Although the KPB also considers only the fastest four machines for each task for the particular value of *k* used here (which happen to be the same four machines for half of the tasks), the performance does not differ much from the inconsistent HiHi case. Additional experiments have shown that the KPB performance is quite insensitive to values of *k* as long as *k* is larger than the minimum value (where the KPB heuristic is reduced to the MET heuristic). For example, when *k* is doubled from its minimum value of 5, the makespan decreases by a factor of about 5. However a further doubling of *k* brings down the makespan by a factor of only about 1.2.

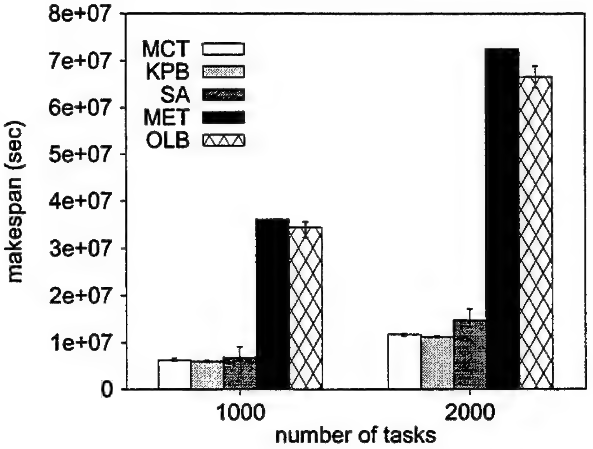


Figure 6. Makespan of the on-line heuristics for semi-consistent HiHi heterogeneity.

6.3. Comparisons of the batch mode heuristics

Figures 8 and 9 compare the batch mode heuristics based on makespan and average sharing penalty, respectively. In these comparisons, unless otherwise stated, the regular time interval strategy is employed to schedule meta-task mapping events. The time interval is set to 10 seconds. This value was empirically found to optimize makespan over other values. From Figure 8, it can be noted that the Sufferage heuristic outperforms the Min-min and the Max-min heuristics based on makespan (although, it is only slightly better than the Min-min). However, for average sharing penalty, the Min-min heuristic outperforms the other heuristics (Figure 9). The Sufferage heuristic considers the "loss" in completion time of a task if it is not assigned to its first choice, in making the mapping decisions. By assigning their first choice machines to the tasks that have the highest sufferage values among all contending tasks, the Sufferage heuristic reduces the overall completion time.

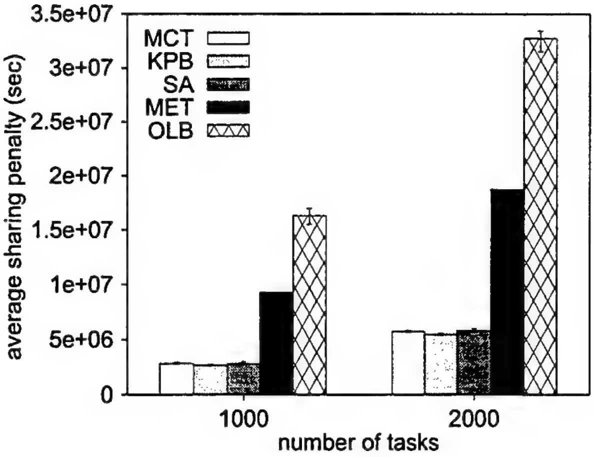


Figure 7. Average sharing penalty of the on-line heuristics for semi-consistent HiHi heterogeneity.

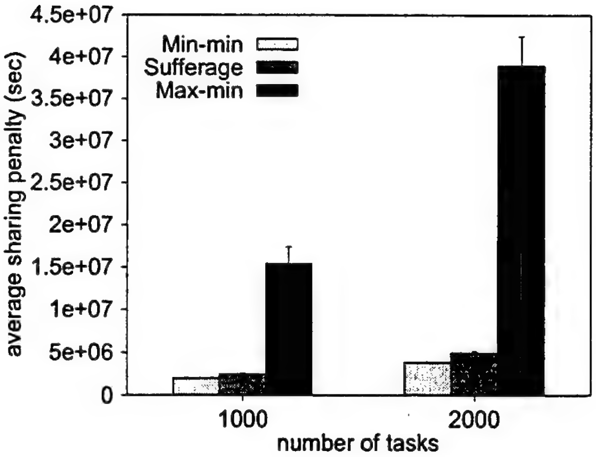


Figure 9. Average sharing penalty of the batch heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity.

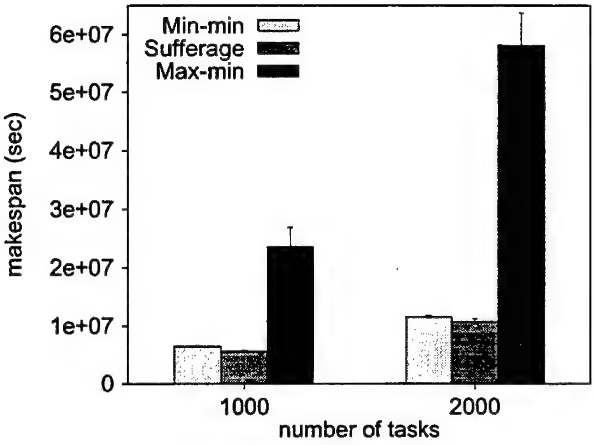


Figure 8. Makespan of the batch heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity.

Furthermore, it can be noted that the makespan given by the Max-min is much larger than the makespans obtained by the other two heuristics. Using reasoning similar to that given in Subsection 4.3 for explaining better expected performance for the Min-min, it can be seen that the Max-min assignments change the machine ready time status by a larger amount than the Min-min assignments do. (The Sufferage also does not necessarily schedule the tasks that finish later first.) If tasks t_i and t_k are contending for a particular machine m_j, then the Max-min assigns m_j to the task (say t_i) that will increase the ready time of m_j more. This decreases the probability that t_k will still have its earliest completion time on m_j and shall be assigned to it. In general, the percentage of tasks assigned their first choice is likely to be lower for the Max-min than for other batch mode heuristics. It might be expected that a larger makespan will result if a larger number of tasks is assigned to the machines that do not have the best execution times for those tasks.

Figure 10 compares the makespan of the batch mode heuristics for semi-consistent HiHi heterogeneity. The comparison of the same heuristics for the same parameters is shown in Figure 11 with respect to average sharing penalty. Results for both average sharing penalty and makespan for semi-consistent HiHi are similar to those for inconsistent HiHi.

The impact of aging on batch mode heuristics is shown in Figures 12 and 13. From Figures 12 and 13, three observations are in order. First, the Max-min heuristic benefits most from the aging scheme. Second, the makespan and the average sharing penalty given by the Sufferage heuristic change negligibly when aging scheme is applied. Third, even though aging schemes are meant to reduce starvation of tasks (as gauged by average sharing penalty), they also reduce the makespan.

The fact that the Max-min benefits most from the aging scheme can be explained using the reasoning given in the discussion on starvation in Subsection 4.3. The larger the number (say N_{new}) of newly arriving tasks between the mapping events τ_i and τ_{i+1}, the larger the probability that some of the tasks mapped at mapping event τ_i, or earlier, will be starved (due to more competing tasks). The Max-min heuristic schedules tasks that finish later first. As mapping events are not scheduled if machines are busy, two successive mapping events in the Max-min are likely to be separated by a larger time duration than those in the Sufferage or the Min-min. The value of N_{new} is therefore likely to be larger in the Max-min schedules, and starvation is more likely to occur. Consequently, aging schemes would make greater difference to the Max-min schedules: the tasks that finish sooner are much more likely to be scheduled before the tasks that finish later in the Max-min with aging than in the Max-min without aging. In contrast to the Max-min (or the Min-min) operation, the Sufferage heuristic optimizes a machine assignment only over the tasks that are contending for that particular machine. This reduces the probability of competition between the "older" tasks and the new arrivals, which in turn reduces the need for an aging scheme, or the improvement in schedule in case aging is implemented.

Figures 14, 15, 16, and 17 show the results of repeating the above experiments with a batch count mapping strategy for a batch size of 40. This particular batch size was found to give an optimum value of the makespan. Figure 14 compares regular time interval strategy and fixed count strategy on the basis of makespans given by different heuristics for inconsistent HiHi heterogeneity. In Figure 15, the average sharing penalties of the same heuristics for the same parameters are compared. It can be seen that the fixed count approach gives essentially the same results for the Min-min and the Sufferage heuristics. The Max-min heuristic, however, benefits considerably from the fixed count approach; makespan drops to about 60% for $| K |= 1000$, and to about 50% for $| K |= 2000$ as compared to the makespan given by the regular time interval strategy. A possible explanation lies in a conceptual element of similarity between the fixed count approach and the aging scheme. A "good" value of κ in fixed count strategy is neither too small to allow only a limited optimization of machine assignment nor too large to subject the tasks carried over from the previous mapping events to a possibly defeating competition with the new or recent arrivals. Figures 16 and 17 show the makespan and the average sharing penalty given for the semi-consistent case. These results show that, for the Sufferage and the Min-min, the regular time interval approach gives slightly better results than the fixed count approach. For the Max-min, however, the fixed count approach gives better performance.

6.4. Comparing on-line and batch heuristics

In Figure 18, two on-line mode heuristics, the MCT and the KPB, are compared with two batch mode heuristics, the Min-min and the Sufferage. The comparison is performed with Poisson arrival rate set to λ_h. It can be noted that for the higher arrival rate and larger $| K |$, batch heuristics are superior to on-line heuristics. This is because the number of tasks waiting to begin execution is likely to be larger in above circumstances than in any other, which in turn means that rescheduling is likely to improve many more mappings in such a system. The on-line heuristics consider only one task when they try to optimize machine assignment, and do not reschedule. Recall that the mapping heuristics use a combination of expected and actual task execution times to compute machine ready times. The on-line heuristics are likely to approach the performance of the batch ones at low task arrival rates, because then both classes of heuristics have comparable information about the actual execution times of the tasks. For example, at a certain low arrival rate, the 100-th arriving task might find that 70 previously arrived tasks have completed. At a higher arrival rate, only 20 tasks might have completed when the 100-th task arrived. The above observation is borne out in Figure 19, which shows that the relative performance difference between on-line and batch heuristics decreases with a decrease in arrival rate. Given the observation that the KPB and the Sufferage per-

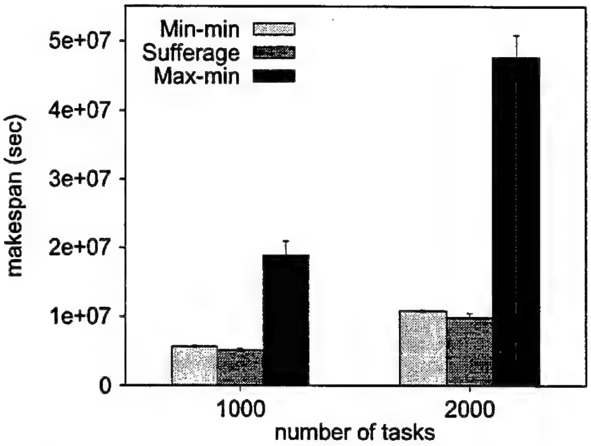


Figure 10. Makespan of the batch heuristics for the regular time interval strategy and semi-consistent HiHi heterogeneity.

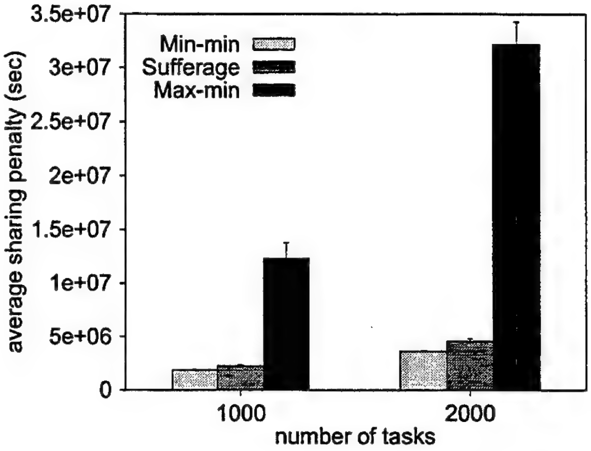


Figure 11. Average sharing penalty of the batch heuristics for the regular time interval strategy and semi-consistent HiHi heterogeneity.

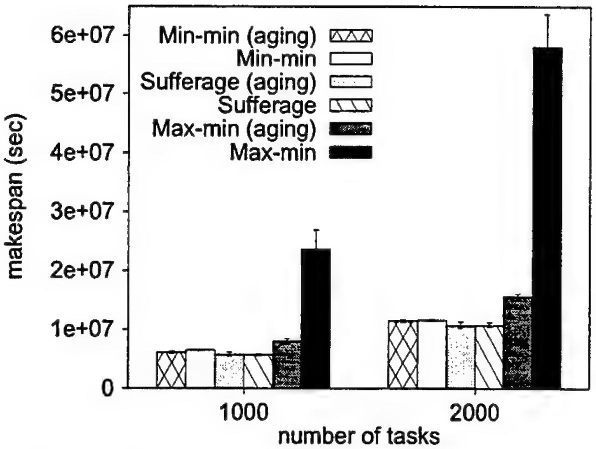


Figure 12. Makespan for the batch heuristics for the regular time interval strategy with and without aging for inconsistent HiHi heterogeneity.

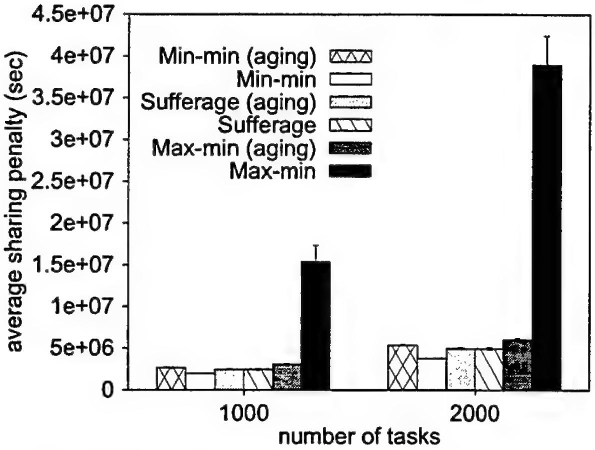


Figure 13. Average sharing penalty of the batch heuristics for the regular time interval strategy with and without aging for inconsistent HiHi heterogeneity.

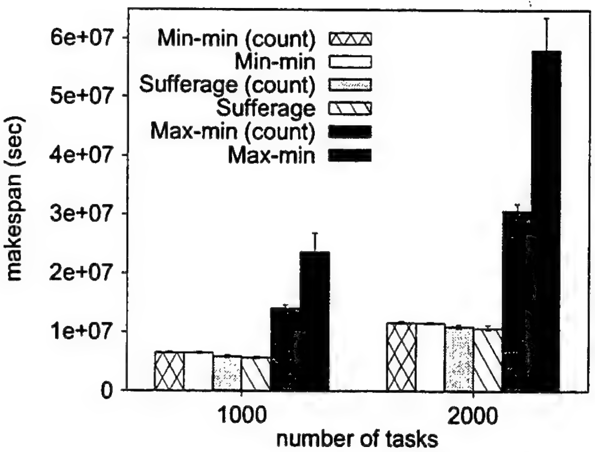


Figure 14. Comparison of the makespans given by the fixed count mapping strategy and the regular time interval strategy for inconsistent HiHi heterogeneity.

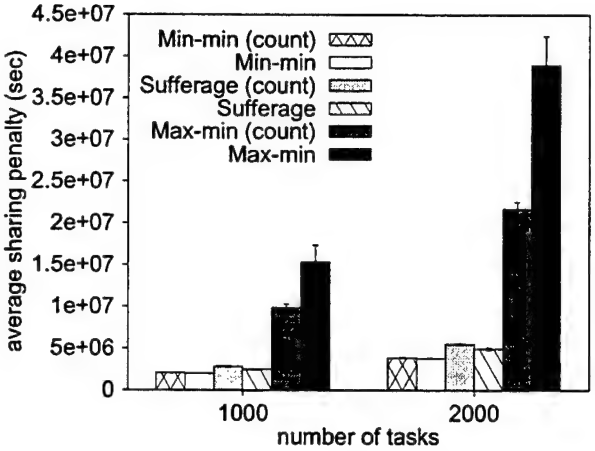


Figure 15. Comparison of the average sharing penalty given by the fixed count mapping strategy and the regular time interval strategy for inconsistent HiHi heterogeneity.

form almost similarly at this low arrival rate, it might be better to use the KPB heuristic because of its smaller computation time. Moreover, Figures 18 and 19 show that the makespan values for all heuristics are larger for lower arrival rate. This is attributable to the fact that at lower arrival rates, a larger fraction of a task's completion time is determined by its beginning time.

7. Conclusions

New and previously proposed dynamic matching and scheduling heuristics for mapping independent tasks onto HC systems were compared under a variety of simulated computational environments. Five on-line mode heuristics and three batch mode heuristics were studied.

In the on-line mode, for both the semi-consistent and the inconsistent types of HiHi heterogeneity, the KPB heuristic outperformed the other heuristics on all performance metrics (however, the KPB was only slightly better than the MCT). The average sharing penalty gains were smaller than the makespan ones. The KPB can provide good system oriented performance (e.g., minimum makespan) and at the same time provide good application oriented performance (e.g., low average sharing penalty). The relative performance of the OLB and the MET with respect to the makespan reversed when the heterogeneity was changed

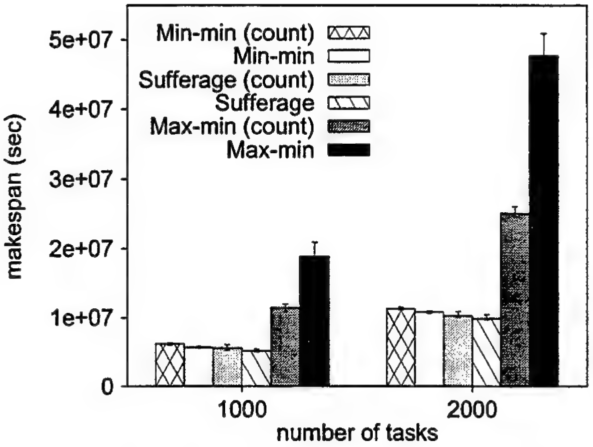


Figure 16. Comparison of the makespan given by the fixed count mapping strategy and the regular time interval strategy for semi-consistent HiHi heterogeneity.

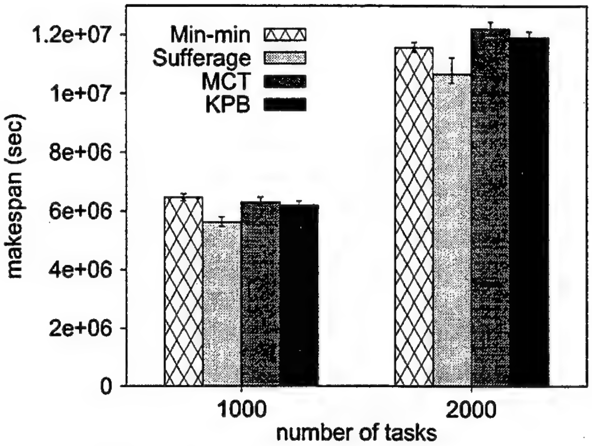


Figure 18. Comparison of the makespan given by batch heuristics (regular time interval strategy) and on-line heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_h.

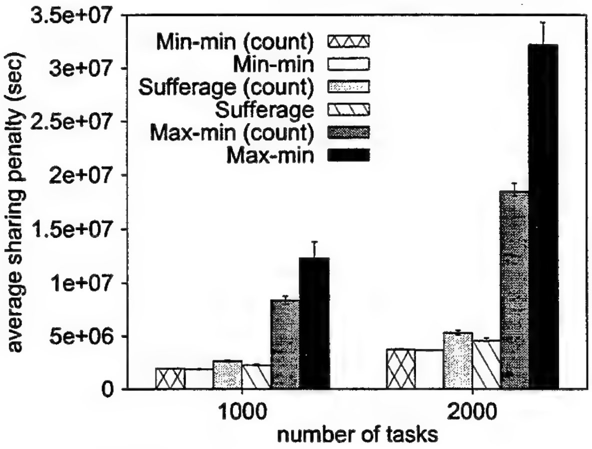


Figure 17. Comparison of the average sharing penalty given by the fixed count mapping strategy and the regular time interval strategy for semi-consistent HiHi heterogeneity.

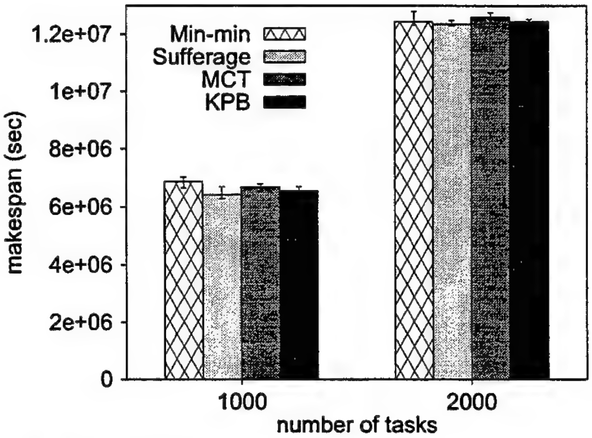


Figure 19. Comparison of the makespan given by batch heuristics (regular time interval strategy) and on-line heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_l.

from the semi-consistent to the inconsistent. The OLB did better than the MET for the semi-consistent case.

In the batch mode, for the semi-consistent and the inconsistent types of HiHi heterogeneity, the Min-min heuristic outperformed the Sufferage and Max-min heuristics in the average sharing penalty. However, the Sufferage performed the best with respect to makespan for both the semi-consistent and the inconsistent types of HiHi heterogeneity (though, the Sufferage was only slightly better than the Min-min).

The batch heuristics are likely to give a smaller makespan than the on-line ones for large $| K |$ and high task arrival rate. For smaller values of $| K |$ and lower task arrival rates, the improvement in makespan offered by batch heuristics is likely to be nominal.

This study quantifies how the relative performance of these dynamic mapping heuristics depends on (a) the consistency property of the ETC matrix, (b) the requirement to optimize system oriented or application oriented performance metrics (e.g., optimizing makespan versus optimizing average sharing penalty), and (c) the arrival rate of the tasks. Thus, the choice of the heuristic which is best to use will be a function of such factors. Therefore, it is important to include a set of heuristics in a resource management system for HC environments, and then use the heuristic that is most appropriate for a given situation (as will be done in the Scheduling Advisor for MSHN).

Acknowledgments: The authors thank Taylor Kidd, Surjamukhi Chatterjea, and Tracy D. Braun for their valuable comments and suggestions.

References

[1] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predications," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79–87.

[2] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance*, Master's thesis, Department of Computer Science, Naval Postgraduate School, 1997 (D. Hensgen, advisor).

[3] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1998, pp. 330–335 (included in the proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, 1998).

[4] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, R. F. Freund, and D. Hensgen, "A comparison study of static mapping heuristics for a class of metatasks on heterogeneous computing systems," *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, Apr. 1999, to appear.

[5] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Fransisco, CA, 1999.

[6] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184–199.

[7] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[8] D. Hensgen, T. Kidd, M. C. Schnaidt, D. St. John, H. J. Siegel, T. D. Braun, M. Maheshwaran, S. Ali, J-K. Kim, C. Irvine, T. Levin, R. Wright, R. F. Freund, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: A Management System for Heterogeneous Networks," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, to appear.

[9] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[10] M. A. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 70–78.

[11] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July-Sep. 1998, pp. 42–51.

[12] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th IEEE Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30–34.

[13] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, *A Comparison of Dynamic Strategies for Mapping a Class of Independent Tasks onto Heterogeneous Computing Systems*, Technical Report, School of Electrical and Computer Engineering, Purdue University, in preparation, 1999.

[14] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., John Wiley, New York, NY, scheduled to appear in 1999.

[15] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Adaptive load sharing in heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, Vol. 9, No. 4, Aug. 1990, pp. 331–346.

[16] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, NY, 1984.

[17] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[18] H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems," *IEE Proceedings on Computer and Digital Techniques*, Vol. 141, No. 1, Jan. 1994, pp. 1–10.

[19] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[20] V. Suresh and D. Chaudhuri, "Dynamic rescheduling–A survey of research," *International Journal of Pro-*

duction Economics, Vol. 32, No. 1, Aug. 1993, pp. 53–63.

[21] P. Tang, P. C. Yew, and C. Zhu, "Impact of self-scheduling on performance of multiprocessor systems," *3rd International Conference on Supercomputing*, July 1988, pp. 593–603.

Biographies

Muthucumaru Maheswaran is an Assistant Professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a BSc degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an MSEE degree in 1994 and a PhD degree in 1998, both from the School of Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an MSEE student at Purdue University. His research interests include computer architecture, distributed computing, heterogeneous computing, Internet and world wide web systems, metacomputing, mobile programs, network computing, parallel computing, resource management systems for metacomputing, and scientific computing. He has authored or coauthored 15 technical papers in these and related areas. He is a member of the Eta Kappa Nu honorary society.

Shoukat Ali is pursuing an MSEE degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a Research Assistant. His main research topic is dynamic mapping of meta-tasks in heterogeneous computing systems. He has held teaching positions at Aitchison College and Keynesian Institute of Management and Sciences, both in Lahore, Pakistan. He was also a Teaching Assistant at Purdue. Shoukat received his BS degree in electrical and electronic engineering from the University of Engineering and Technology, Lahore, Pakistan in 1996. His research interests include computer architecture, parallel computing, and heterogeneous computing.

Howard Jay Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received BS degrees in both electrical engineering and management from MIT, and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Debra Hensgen is an Associate Professor in the Computer Science Department at The Naval Postgraduate School. She received her PhD in the area of Distributed Operating Systems from the University of Kentucky. She is currently a Principal Investigator of the DARPA-sponsored Management System for Heterogeneous Networks QUORUM project (MSHN) and a co-investigator of the DARPA-sponsored Server and Active Agent Management (SAAM) Next Generation Internet project. Her areas of interest include active modeling in resource management systems, network re-routing to preserve quality of service guarantees, visualization tools for performance debugging of parallel and distributed systems, and methods for aggregating sensor information. She has published numerous papers concerning her contributions to the Concurra toolkit for automatically generating safe, efficient concurrent code, the Graze parallel processing performance debugger, the SAAM path information base, and the SmartNet and MSHN Resource Management Systems.

Richard F. Freund is a founder and CEO of NOEMIX, a San Diego based startup to commercialize distributed computing technology. Freund is also one of the early pioneers in the field of distributed computing, in which he has written or co-authored a number of papers. In addition he is a founder of the Heterogeneous Computing Workshop, held each year in conjunction with IPPS/SPDP. Freund won a Meritorious Civilian Service Award during his former career as a government scientist.

Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems

Muthucumaru Maheswaran[‡], *Shoukat Ali*[†], *Howard Jay Siegel*[†],
Debra Hensgen[◇], *and Richard F. Freund*[*]

[‡]Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2 Canada
Email: maheswar@cs.umanitoba.ca

[†]Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
Email: {alis,hj}@ecn.purdue.edu

[◇]Department of Computer Science
Naval Postgraduate School
Monterey, CA 93940 USA
Email: hensgen@cs.nps.navy.mil

[*]NOEMIX Inc.
1425 Russ Blvd. Ste. T-110
San Diego, CA 92101 USA
Email: noemix@home.com

June 1999

Accepted to the Journal of Parallel and Distributed Computing
Special Issue on Software Support for Distributed Computing

This research was supported by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-97-M-0900, N62271-98-M-0217, and N62271-98-M-0448, and under the GSA subcontract number GS09K99BH0250. Some of the equipment used was donated by Intel and Microsoft.

Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous...

Corresponding Author:

Shoukat Ali
Mailbox 64
1285 School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285 USA
Phone: 765 494 3750
Fax: 765 494 2706
Email: alis@ecn.purdue.edu

Abstract

Dynamic mapping (matching and scheduling) heuristics for a class of independent tasks using heterogeneous distributed computing systems are studied. Two types of mapping heuristics are considered: immediate mode and batch mode heuristics. Three new heuristics, one for batch mode and two for immediate mode, are introduced as part of this research. Simulation studies are performed to compare these heuristics with some existing ones. In total, five immediate mode heuristics and three batch mode heuristics are examined. The immediate mode dynamic heuristics consider, to varying degrees and in different ways, task affinity for different machines and machine ready times. The batch mode dynamic heuristics consider these factors, as well as aging of tasks waiting to execute. The simulation results reveal that the choice of which dynamic mapping heuristic to use in a given heterogeneous environment depends on parameters such as: (a) the structure of the heterogeneity among tasks and machines, and (b) the arrival rate of the tasks.

Keywords: batch mode mapping, dynamic mapping, mapping heuristics, meta-task mapping, immediate mode mapping.

1. Introduction

In general, heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize their combined performance and/or cost-effectiveness [6, 9, 18]. HC is an important technique for efficiently solving collections of computationally intensive problems [7]. As machine architectures become more advanced to obtain higher peak performance, the extent to which a given task can exploit a given architectural feature depends on how well the task's computational requirements match the machine's advanced capabilities. The applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources. HC systems have resource management systems (RMSs) to govern the execution of the tasks that arrive for service. This paper describes and compares eight heuristics that can be used in such an RMS for dynamically assigning independent tasks to machines.

In a general HC system, schemes are necessary to assign tasks to machines (matching), and to compute the execution order of the tasks assigned to each machine (scheduling) [3]. The process of matching and scheduling tasks is referred to as mapping. Dynamic methods to do this operate on-line, i.e., as tasks arrive. This is in contrast to static techniques, where the complete set of tasks to be mapped is known *a priori*, the mapping is done off-line, i.e., prior to the execution of any of the tasks, and more time is available to compute the mapping (e.g., [4, 27].

In the HC environment considered here, the tasks are assumed to be independent, i.e., no communications between the tasks are needed. This scenario is likely to be present, for instance, when many independent users submit their jobs to a collection of shared computational resources. A dynamic scheme is needed because the arrival times of the tasks may be random and some machines in the suite may go off-line and new machines may come on-line. The dynamic mapping heuristics investigated in this study are non-preemptive, and assume that the tasks have no deadlines or priorities associated with them.

The mapping heuristics can be grouped into two categories: immediate mode and batch mode heuristics. In the immediate mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The independent set of tasks that is considered for mapping at the mapping events is called a meta-task. A meta-task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and the ones that were mapped in earlier mapping events but did not begin execution. While immediate mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

The trade-offs among and between immediate mode and batch mode heuristics are studied experimentally. Mapping independent tasks onto an HC suite is a well-known NP-complete problem if throughput is the optimization criterion [12]. For the heuristics discussed in this paper, maximization of throughput is the primary objective, because this performance measure is the most common one in production oriented environments.

Three new heuristics, one for batch mode and two for immediate mode, are introduced as part of this research. Simulation studies are performed to compare these heuristics with some existing ones. In total, five immediate mode heuristics and three batch mode heuristics are examined. The immediate mode heuristics consider, to varying degrees and in different ways, task affinity for different machines and machine ready times. The batch mode heuristics consider these factors, as well as aging of tasks waiting to execute.

Section 2 describes some related work. Section 3 defines an optimization criterion and discusses the mapping approaches studied here. The simulation procedure is given in Section 4. Section 5 presents the simulation results.

This research is part of a DARPA/ITO Quorum Program project called MSHN (pronounced "mission") (Management System for Heterogeneous Networks) [11]. MSHN is a collaborative research effort that includes the Naval Postgraduate School, NOEMIX, Purdue, and University of Southern California. It builds on SmartNet, an implemented scheduling framework and system for managing resources in an HC environment developed at NRaD [8]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service. The heuristics developed here, or their derivatives, may be included in the Scheduling Advisor component of the MSHN prototype.

2. Related Work

Related work in literature was examined to select a set of heuristics appropriate for the HC environment considered here, and then perform comparative studies. This section is a sampling of related literature, and is not meant to be exhaustive.

In the literature, mapping tasks onto machines is often referred to as scheduling. Several researchers have worked on the dynamic mapping problem from areas including job shop scheduling and distributed computer systems (e.g., [13, 16, 23, 25]).

The heuristics presented in [12] are concerned with mapping independent tasks onto heterogeneous machines such that the completion time of the last finishing task is minimized. The problem is recognized as NP-complete, and worst case performance bounds are obtained for the heuristics. Some of these heuristics are designed for a general HC environment, while the rest target either a heterogeneous two machine system or a general homogeneous system. Of the heuristics designed for a general HC environment, the A-schedule, B-schedule, and C-schedule heuristics are variations of the minimum completion time heuristic used here. The Min-min heuristic that is used here as a benchmark for batch mode mapping is based on the D-schedule, and is also one of the heuristics implemented in SmartNet [8].

The scheme in [13] is representative of techniques for mapping communicating subtasks to an HC suite, considering data dependency graphs and communication times between machines. Thus, an environment very different than the set of independent tasks considered here is used. Hence, the heuristics developed for that different environment are not appropriate for the HC environment considered here.

Two dynamic mapping approaches, one using a distributed policy and the other using a centralized policy, are developed in [16]. Both of these approaches are very similar to the minimum completion time heuristic (used as a benchmark in the studies here) except that they incorporate communication times in calculating the minimum completion time for a task. For the distributed approach, the mapper at a given node considers the local machine and the k highest communication bandwidth neighbors to map the tasks in the local queue. Therefore, the mapper based on the distributed strategy assigns a task to the best machine among the $k+1$ machines. The simulation results provided in [16] show that the heuristic with the centralized policy always performs better than the distributed heuristic. Hence, the minimum completion time heuristic used here represents the better of the two heuristics presented in [16].

A survey of dynamic scheduling heuristics for job-shop environments is provided in [25]. It classifies the dynamic scheduling algorithms into three approaches: knowledge-based approach, conventional approach, and distributed problem solving approach. The heuristics with a knowledge-based approach take a long time to execute, and hence are not suitable for the particular dynamic environment considered here. The classes of heuristics grouped under the conventional and distributed problem solving approaches are similar to the minimum completion time heuristic considered in this paper. However, the problem domains considered in [25] involve precedence constraints among the tasks, priorities, or deadlines, and thus differ from the domain here.

In distributed computer systems, load balancing schemes have been a popular strategy for mapping tasks onto machines (e.g., [19, 23]). In [19], the performance characteristics of simple load balancing heuristics for HC distributed systems are studied. The heuristics presented in [19] do not consider task execution times when making their decisions. In [23], a survey of dynamic scheduling heuristics for distributed computing systems is provided. All heuristics, except one, in [23] schedule tasks on different machines using load sharing techniques, without considering task execution times. (The one heuristic in [23] that does not use load sharing, employs deadlines to schedule tasks, and therefore falls out of the problem domain discussed here.) The load balancing heuristic used in this research is representative of the load balancing techniques in [19] and [23].

SmartNet [8] is an RMS for HC systems that employs various heuristics to map tasks to machines considering resource and task heterogeneity. In this paper, some SmartNet heuristics appropriate for the HC environment considered here are included in the comparative study (minimum completion time, Min-min, and Max-min).

3. Mapping Heuristics

3.1. Overview

The expected execution time e_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given m_j has no load when t_i is assigned. The time e_{ij} includes the time to move the t_i code and data from each of their corresponding single fixed sources to machine m_j. The expected completion time c_{ij} of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let m be the total number of machines in the HC suite. Let K be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be a_i, and let the time t_i begins execution be b_i. From the above definitions, $c_{ij} = b_i + e_{ij}$. Let c_i be the completion time for task t_i, and is equal to c_{ij} where machine m_j is assigned to execute task t_i. The makespan [21] for the complete schedule is then defined as $max_{t_i \in K}(c_i)$. Makespan is a measure of the throughput of the HC system, and does not measure the quality of service imparted to an individual task. One other performance metric is considered in [17].

In the immediate mode heuristics, each task is considered only once for matching and scheduling, i.e., the mapping is not changed once it is computed. When the arrival rate is low enough, machines may be ready to execute a task as soon as it arrives at the mapper. Therefore, it may be beneficial to use the mapper in the immediate mode so that a task need not wait until the next mapping event to begin its execution.

Recall from Section 1, in immediate mode the mapper assigns a task to a machine as soon as the task arrives at the mapper, and in batch mode a set of independent tasks that needs to be mapped at a mapping event is called a meta-task. (In some systems, the term meta-task is defined in a way that allows inter-task dependencies.) In batch mode, for the i-th mapping event, the meta-task M_i is mapped at time τ_i, where $i \geq 0$. The initial meta-task, M_0, consists of all the tasks that arrived prior to time τ_0, i.e., $M_0 = \{t_j \mid a_j < \tau_0\}$. The meta-task, M_k, for $k > 0$, consists of tasks that arrived after the last mapping event and the tasks that had been mapped, but did not start executing, i.e., $M_k = \{t_j \mid \tau_{k-1} \leq a_j < \tau_k\} \cup \{t_j \mid a_j < \tau_{k-1}, b_j > \tau_k\}$.

In batch mode, the mapper considers a meta-task for matching and scheduling at each mapping event. This enables the mapping heuristics to possibly make better decisions than immediate mode heuristics. This is because the batch mode heuristics have the resource requirement information for a whole meta-task, and know about the actual execution times of a larger number of tasks (as more tasks might complete while waiting for the mapping event). When the task arrival rate is high, there will be a sufficient number of tasks to keep the machines busy in between the mapping events, and while a mapping is being computed. (It is, however, assumed in this study that the running time of each mapping heuristic is negligibly small as compared to the average task execution time.)

Both immediate mode and batch mode heuristics assume that estimates of expected task execution times on each machine in the HC suite are known. The assumption that these estimated expected times are known is commonly made when studying mapping heuristics for HC systems (e.g., [10, 15, 24]). (Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [18].) These estimates can be supplied before a task is submitted for execution, or at the time it is submitted.

The ready time of a machine is the earliest wall clock time that machine is going to be ready after completing the execution of the tasks that are currently assigned to it. Because the heuristics presented here are dynamic, the expected machine ready times are based on a combination of actual task execution times (for tasks that have completed execution on that machine) and estimated expected task execution times (for tasks assigned to that machine and waiting to execute). It is assumed that each time a task t_i completes on a machine m_j a report is sent to the mapper, and the ready time for m_j is updated if necessary. The experiments presented in Section 5 model this situation using simulated actual values for the execution times of the tasks that have already finished their execution.

All heuristics examined here operate in a centralized fashion and map tasks onto a dedicated suite of machines; i.e., the mapper controls the execution of all jobs on all machines in the suite. It is assumed that each mapping heuristic is being run on a separate machine. (While all heuristics studied here are functioning dynamically, the use of some of these heuristics in a static environment is discussed in [4].)

3.2. Immediate mode mapping heuristics

Five immediate mode heuristics are described here. These are (i) minimum completion time, (ii) minimum execution time, (iii) switching algorithm, (iv) k-percent best, and (v) opportunistic load balancing. Of these five heuristics, switching algorithm and k-percent best have been proposed as part of the research presented here.

The minimum completion time (MCT) heuristic assigns each task to the machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The MCT heuristic is a variant of the fast-greedy heuristic from SmartNet [8]. The MCT heuristic is used as a benchmark for the immediate mode, i.e., the performance of the other heuristics is compared against that of the MCT heuristic. As a task arrives, all the machines in the HC suite are examined to determine the machine that gives the earliest completion time for the task. Therefore, it takes $O(m)$ time to map a given task.

The minimum execution time (MET) heuristic assigns each task to the machine that performs that task's computation in the least amount of execution time (this heuristic is also known as limited best assignment (LBA) [1] and user directed assignment (UDA) [8]). This heuristic, in contrast to MCT, does not consider machine ready times. This heuristic can cause a severe imbalance in load across the machines. The advantages of this method are that it gives each task to the machine that performs it in the least amount of execution time, and the heuristic is very simple. The heuristic needs $O(m)$ time to find the machine that has the minimum execution time for a task.

The switching algorithm (SA) is motivated by the following observations. The MET heuristic can potentially create load imbalance across machines by assigning many more tasks to some machines than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. If the tasks are arriving in a random mix, it is possible to use the MET at the expense of load balance until a given threshold and then use the MCT to smooth the load across the machines. The SA heuristic uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. The purpose is to have a heuristic with the desirable properties of both the MCT and the MET.

Let the maximum (latest) ready time over all machines in the suite be $\underline{r_{max}}$, and the minimum (earliest) ready time be $\underline{r_{min}}$. Then, the <u>load</u> <u>balance</u> <u>index</u> across the machines is given by $\underline{\pi} = r_{min}/r_{max}$. The parameter π can have any value in the interval $[0,1]$. If π is 1.0, then the load is evenly balanced across the machines. If π is 0, then at least one machine has not yet been assigned a task. Two threshold values, $\underline{\pi_l}$ (low) and $\underline{\pi_h}$ (high), for the ratio π are chosen in $[0,1]$ such that $\pi_l < \pi_h$. Initially, the value of π is set to 0.0. The SA heuristic begins mapping tasks using the MCT heuristic until the value of load balance index increases to at least π_h. After that point in time, the SA heuristic begins using the MET heuristic to perform task mapping. This typically causes the load balance index to decrease. When it decreases to π_l or less, the SA heuristic switches back to using the MCT heuristic for mapping the tasks and the cycle continues.

As an example of the functioning of the SA with lower and upper limits of 0.6 and 0.9, respectively, for $|K| = 1000$ and one particular rate of arrival of tasks, the SA switched between the MET and the MCT two times (i.e., from the MCT to the MET to the MCT), assigning 715 tasks using the MCT. For $|K| = 2000$ and the same task arrival rate, the SA switched five times, using the MCT to assign 1080 tasks. The percentage of tasks assigned using MCT gets progressively smaller for larger $|K|$. This is because the larger the $|K|$, the larger the number of tasks waiting to execute on a given machine, and therefore, the larger the ready time of a given machine. This in turn means that an arriving task's execution time will impact the machine ready time less, thereby rendering the load balance index less sensitive to a load-imbalancing assignment.

At each task's arrival, the SA heuristic determines the load balance index. In the worst case, this takes $O(m)$ time. In the next step, the time taken to assign a task to a machine is of order $O(m)$ whether SA uses the MET to perform the mapping or the MCT. Overall, the SA heuristic takes $O(m)$ time irrespective of which heuristic is actually used for mapping the task.

The k-percent best (KPB) heuristic considers only a subset of machines while mapping a task. The subset is formed by picking the $m \times (k/100)$ best machines based on the execution times for the task, where $100/m \leq k \leq 100$. The task is assigned to a machine that provides the earliest completion time in the subset. If $k = 100$, then the KPB heuristic is reduced to the MCT heuristic. If $k = 100/m$, then the KPB heuristic is reduced to the MET heuristic. A "good" value of k maps a task to a machine only within a subset formed from computationally superior machines. The purpose is not as much as matching of the current task to a computationally well-matched machine as it is to avoid putting the current task onto a machine which might be more suitable for some yet-to-arrive tasks. This "foresight" about task heterogeneity is missing in the MCT, which might assign a task to a poorly matched machine for a local marginal improvement in completion time, possibly depriving some subsequently arriving better matched tasks of that machine, and eventually leading to a larger makespan as compared to the KPB. It should be noted that while both the KPB and SA combine elements of the MCT and the MET in their operation, it is only in the KPB that *each* task assignment attempts to optimize objectives of the MCT and the MET simultaneously. However, in cases where a fixed subset of machines is not among the k% best for any of the tasks, the KPB will cause more machine idle time compared to the MCT, and can result in much poorer performance. Thus the relative performance of the KPB and the MCT may depend on the HC suite of machines, and characteristics of the tasks being executed.

For each task, $O(m \log m)$ time is spent in ranking the machines for determining the subset of machines

Table 1: Initial ready times of the machines (arbitrary units).

m_0	m_1	m_2
75	110	200

to examine. Once the subset of machines is determined, it takes $O(\frac{m \times k}{100})$ time, i.e., $O(m)$ time to determine the machine assignment. Overall the KPB heuristic takes $O(m \log m)$ time.

The opportunistic load balancing (OLB) heuristic assigns a task to the machine that becomes ready next, without considering the execution time of the task onto that machine. If multiple machines become ready at the same time, then one machine is arbitrarily chosen. The complexity of the OLB heuristic is dependent on the implementation. In the implementation considered here, the mapper may need to examine all m machines to find the machine that becomes ready next. Therefore, it takes $O(m)$ to find the assignment. Other implementations may require idle machines to assign tasks to themselves by accessing a shared global queue of tasks [26].

As an example of the working of these heuristics, consider a simple system of three machines, m_0, m_1, and m_2, currently loaded so that expected ready times are as given in Table 1. Consider the performance of the heuristics for a very simple case of three tasks t_0, t_1, and t_2 arriving in that order. Table 2 shows the expected execution times of tasks on the machines in the system. All time values in the discussion below are the expected values.

The MET finds that all tasks have their minimum completion time on m_2, and even though m_2 is already heavily loaded, it assigns all three tasks to m_2. The time when t_0, t_1, and t_2 will all have completed is 245 units.

The OLB assigns t_0 to m_0 because m_0 is expected to be idle soonest. Similarly, it assigns t_1 and t_2 to m_1

Table 2: Expected execution times (arbitrary units).

	m_0	m_1	m_2
t_0	50	20	15
t_1	20	60	15
t_2	20	50	15

and m_0, respectively. The time when t_0, t_1, and t_2 will all have completed is 170 units.

The MCT determines that the minimum completion time for t_0 will be achieved on m_0, and makes this assignment, even though the execution time of t_0 on m_0 is more than twice that on m_1 (where the completion time would have been only slightly larger). Then MCT goes on to assign t_1 to m_0, and t_2 to m_1 so that the time when t_0, t_1, and t_2 will all have completed is 160 units.

The SA first determines the current value of the load balance index, π, which is $75/200$ or 0.38. Assume that π_l is 0.40 and that π_h is 0.70. Because $\pi < \pi_l$, the SA chooses the MCT to make the first assignment. After the first assignment, $\pi = 110/200 = 0.55 < \pi_h$. So the SA continues to use the MCT for the second assignment as well. It is only after third assignment that $\pi = 145/200 = 0.725 > \pi_h$ so that the SA will then use the MET for the fourth arriving task. The time when t_0, t_1, and t_2 will all have completed here is the same as that for the MCT.

Let the value of k in the KPB be 67% so that the KPB will choose from the two fastest executing machines to assign a given task. For t_0, these machines are m_1 and m_2. Within these two machines, the minimum completion time is achieved on m_1 so t_0 is assigned to m_1. This is the major difference from the working of the MCT; m_0 is not assigned t_0 even though t_0 would have its minimum completion time (over all machines) there. This step saves m_0 for any yet-to-arrive tasks that may run slowly on other machines. One such task is t_2; in the MCT it is assigned to m_1, but in the KPB it is assigned to m_0. The time when t_0,

t_1, and t_2 will all have completed using the KPB is 130 units. This is the smallest among all five heuristics.

3.3. Batch mode mapping heuristics

Three batch mode heuristics are described here: (i) the Min-min heuristic, (ii) the Max-min heuristic, and (iii) the Sufferage heuristic. The Sufferage heuristic has been proposed as part of the research presented here. In the batch mode heuristics, meta-tasks are mapped after predefined intervals. These intervals are defined in this study using one of the two strategies proposed below.

The regular time interval strategy maps the meta-tasks at regular intervals of time (e.g., every ten seconds). The only occasion when such a mapping will be redundant is when: (1) no new tasks have arrived since the last mapping, and (2) no tasks have finished executing since the last mapping (thus, machine ready times are unchanged). These conditions can be checked for, and so redundant mapping events can be avoided.

The fixed count strategy maps a meta-task M_i as soon as one of the following two mutually exclusive conditions are met: (a) an arriving task makes $| M_i |$ larger than or equal to a predetermined arbitrary number κ, or (b) all tasks in the set $| K |$ have arrived, and a task completes while the number of tasks which yet have to begin is larger than or equal to κ. In this strategy, the time between the mapping events will depend on the arrival rate and the completion rate. The possibility of machines being idle while waiting for the next mapping event will depend on the arrival rate, completion rate, m, and κ. (For the arrival rates in the experiments here, this strategy operates reasonably; in an actual system, it may be necessary for tasks to have a maximum waiting time to be mapped.)

The batch mode heuristics considered in this study are discussed in the paragraphs below. The complexity analysis performed for these heuristics considers a single mapping event, and the meta-task size is assumed to be equal to the average of meta-task sizes at all actually performed mapping events. Let the average meta-task size be S.

The Min-min heuristic shown in Figure 1 is from [12], and is one of the heuristics implemented in SmartNet [8]. In Figure 1, let r_j denote the expected time machine m_j will become ready to execute a task after finishing the execution of all tasks assigned to it at that point in time. First the c_{ij} entries are computed using the e_{ij} and r_j values. For each task t_i, the machine that gives the earliest expected completion time is determined by scanning the i-th row of the c matrix (composed of the c_{ij} values). The task t_k that has the minimum earliest expected completion time is determined and then assigned to the corresponding machine. The matrix c and vector r are updated and the above process is repeated with tasks that have not yet been assigned a machine.

Min-min begins by scheduling the tasks that change the expected machine ready time status by the least amount. If tasks t_i and t_k are contending for a particular machine m_j, then Min-min assigns m_j to the task (say t_i) that will change the ready time of m_j less. This increases the probability that t_k will still have its earliest completion time on m_j, and shall be assigned to it. Because at $t = 0$, the machine which finishes a task earliest is also the one that executes it fastest, and from thereon Min-min heuristic changes machine ready time status by the least amount for every assignment, the percentage of tasks assigned their first choice (on basis of expected execution time) is likely to be higher in Min-min than with the other batch mode heuristics described in this section (this has been verified by examining the simulation study data [17]). The expectation is that a smaller makespan can be obtained if a larger number of tasks is assigned to the machines that not only complete them earliest but also execute them fastest.

The initialization of the c matrix in Line (1) to Line (3) of Figure 1 takes $O(Sm)$ time. The **do** loop of the Min-min heuristic is repeated S times and each iteration takes $O(Sm)$ time. Therefore, the heuristic takes

```
(1)  for all tasks $t_i$ in meta-task $M_v$ (in an arbitrary order)
(2)      for all machines $m_j$ (in a fixed arbitrary order)
(3)          $c_{ij} = e_{ij} + r_j$
(4)  do until all tasks in $M_v$ are mapped
(5)      for each task in $M_v$ find the earliest completion
             time and the machine that obtains it
(6)      find the task $t_k$ with the minimum earliest
             completion time
(7)      assign task $t_k$ to the machine $m_l$ that gives the
(8)          earliest completion time
(9)      delete task $t_k$ from $M_v$
(10)     update $r_l$
(11)     update $c_{il}$ for all $i$
(12) enddo
```

Figure 1: The Min-min heuristic.

$O(S^2m)$ time.

The Max-min heuristic is similar to the Min-min heuristic, and is one of the heuristics implemented in SmartNet [8]. It differs from the Min-min heuristic (given in Figure 1) in that once the machine that provides the earliest completion time is found for every task, the task t_k that has the maximum earliest completion time is determined and then assigned to the corresponding machine. That is, in Line (6) of Figure 1, "minimum" would be changed to "maximum." The Max-min heuristic has the same complexity as the Min-min heuristic.

The Max-min is likely to do better than the Min-min heuristic in cases where there are many more shorter tasks than longer tasks. For example, if there is only one long task, Max-min will execute many short tasks concurrently with the long task. The resulting makespan might just be determined by the execution time of the long task in this case. Min-min, however, first finishes the shorter tasks (which may be more or less evenly distributed over the machines) and then executes the long task, increasing the makespan compared to the Max-min.

The Sufferage heuristic (shown in Figure 2) is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it. Let the sufferage value of a task t_i be the difference between its second earliest completion time (on some machine m_y) and its earliest completion time (on some machine m_x). That is, using m_x will result in the best completion time for t_i, and using m_y the second best.

```
for all tasks $t_k$ in meta-task $M_v$ (in an arbitrary order)
    for all machines $m_j$ (in a fixed arbitrary order)
        $c_{kj} = e_{kj} + r_j$
do until all tasks in $M_v$ are mapped
    mark all machines as unassigned
    for each task $t_k$ in $M_v$ (in a fixed arbitrary order)
        /* for a given execution of the for statement,
        each $t_k$ in $M_v$ is considered only once */
        find machine $m_j$ that gives the earliest
            completion time
        sufferage value = second earliest completion
            time – earliest completion time
        if machine $m_j$ is unassigned
            assign $t_k$ to machine $m_j$, delete $t_k$
            from $M_v$, mark $m_j$ assigned
        else
            if sufferage value of task $t_i$ already
            assigned to $m_j$ is less than the
            sufferage value of task $t_k$
                unassign $t_i$, add $t_i$ back to $M_v$,
                assign $t_k$ to machine $m_j$,
                delete $t_k$ from $M_v$
    endfor
    update the vector $r$ based on the tasks that
        were assigned to the machines
    update the $c$ matrix
enddo
```

Figure 2: The Sufferage heuristic.

The initialization phase in Lines (1) to (3), in Figure 2, is similar to the ones in the Min-min and Max-min heuristics. Initially all machines are marked unassigned. In each iteration of the **for** loop in Lines (6) to (14), pick arbitrarily a task t_k from the meta-task. Find the machine m_j that gives the earliest completion time for task t_k, and tentatively assign m_j to t_k if m_j is unassigned. Mark m_j as assigned, and remove t_k from meta-task. If, however, machine m_j has been previously assigned to a task t_i, choose from t_i and t_k the task that has the higher sufferage value, assign m_j to the chosen task, and remove the chosen task from the meta-task. The unchosen task will not be considered again for this execution of the **for** statement, but shall be considered for the next iteration of the **do** loop beginning on Line (4). When all the iterations of the **for** loop are completed (i.e., when one execution of the **for** statement is completed), update the machine ready time of each machine that is assigned a new task. Perform the next iteration of the **do** loop beginning on Line (4) until all tasks have been mapped.

Table 3 shows a scenario in which the Sufferage will outperform the Min-min. Table 3 shows the expected execution time values for four tasks on four machines (all initially idle). In this case, the Min-min heuristic gives a makespan of 93 and the Sufferage heuristic gives a makespan of 78. Figure 3 gives a pictorial representation of the assignments made for the case in Table 3.

From the pseudo code given in Figure 2, it can be observed that first execution of the **for** statement on Line (6) takes $O(Sm)$ time. The number of task assignments made in one execution of this **for** statement depends on the total number of machines in the HC suite, the number of machines that are being contended for among different tasks, and the number of tasks in the meta-task being mapped. In the worst case, only one task assignment will be made in each execution of the **for** statement. Then meta-task size will decrease by one at each **for** statement execution. The outer **do** loop will be iterated S times to map the whole meta-task. Therefore, in the worst case, the time $T(S)$ taken to map a meta-task of size S will be

$$T(S) = Sm + (S-1)m + (S-2)m + \cdots + m$$

$$T(S) = O(S^2 m)$$

In the best case, there are as many machines as there are tasks in the meta-task, and there is no contention among the tasks. Then all the tasks are assigned in the first execution of the **for** statement so that $T(S) = O(Sm)$. Let ω be a number quantifying the extent of contention among the tasks for the different machines. The complexity of the Sufferage heuristic can then be given as $O(\omega Sm)$, where $1 \leq \omega \leq S$. It can be seen that ω is equal to S in the worst case, and is 1 in the best case; these values of ω are numerically equal to the number of iterations of the **do** loop on Line (4), for the worst and the best case, respectively.

Table 3: An example expected execution time matrix that illustrates the situation where the Sufferage heuristic outperforms the Min-min heuristic.

	m_0	m_1	m_2	m_3
t_0	40	48	134	50
t_1	50	82	88	89
t_2	55	68	94	93
t_3	52	60	78	108

The batch mode heuristics can cause some tasks to be starved of machines. Let H_i be a subset of meta-task M_i consisting of tasks that were mapped (as part of M_i) at the mapping event i at time τ_i but did not begin execution by the next mapping event at τ_{i+1}. H_i is the subset of M_i that is included in M_{i+1}. Due to the expected heterogeneous nature of the tasks, the meta-task M_{i+1} may be mapped so that some or all of the tasks arriving between τ_i and τ_{i+1} may begin executing before the tasks in set H_i do. It is possible that some or all of the tasks in H_i may be included in H_{i+1}. This probability increases as the number of new tasks arriving between τ_i and τ_{i+1} increases. In general, some tasks may be remapped at each successive mapping event without actually beginning execution (i.e., the task is starving for a machine). This impacts the response time the user sees (this is examined as a "sharing penalty" in [17]).

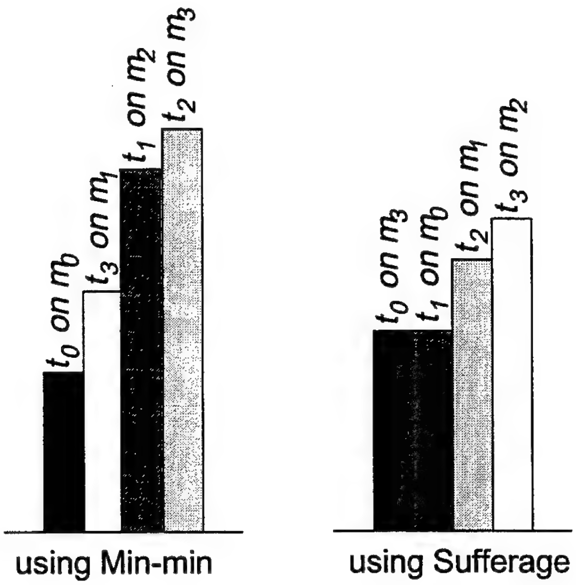


Figure 3: An example scenario (based on Table 3) where the Sufferage gives a shorter makespan than the Min-min (bar heights are proportional to task execution times).

To reduce starvation, aging schemes are implemented. The age of a task is set to zero when it is mapped for the first time and incremented by one each time the task is remapped. Let σ be a constant that can be adjusted empirically to change the extent to which aging affects the operation of the heuristic. An aging factor, $\zeta = (1 + \mathrm{age}/\sigma)$, is then computed for each task. For the experiments in this study, σ is arbitrarily set to 10 (e.g., in this case the aging factor for a task increases by one after every ten remappings of the task). The aging factor is used to enhance the probability of an "older" task beginning before the tasks that would otherwise begin first. In the Min-min heuristic, for each task, the completion time obtained in Line (5) of Figure 1 is multiplied by the corresponding value for $\frac{1}{\zeta}$. As the age of a task increases, its age-compensated expected completion time (i.e., one used to determine the mapping) gets increasingly smaller than its original expected completion time. This increases its probability of being selected in Line (6) in Figure 1.

For the Max-min heuristic, the completion time of a task is multiplied by ζ. In the Sufferage heuristic, the sufferage value computed in Line (8) in Figure 2 is multiplied by ζ.

4. Simulation Procedure

The mappings are simulated using a discrete event simulator (e.g., [5, 14, 22]). The task arrivals are modeled by a Poisson random process. The simulator contains an ETC (expected time to compute) matrix that contains the expected execution times of a task on all machines, for all the tasks that can arrive for service. The ETC matrix entries used in the simulation studies represent the e_{ij} values (in seconds) that the heuristic would use in its operation. The actual execution time of a task can be different than the value given by the ETC matrix. This variation is modeled by generating a simulated actual execution time for each task by sampling a Gaussian probability density function with variance equal to three times the expected execution time of the task and mean equal to the expected execution time of the task (e.g., [2, 20]). If the sampling results in a negative value, the value is discarded and the same probability density function is sampled again (i.e., a truncated Gaussian distribution is sampled). This process is repeated until a positive value is returned by the sampling process.

In an ETC matrix, the numbers along a row indicate the estimated expected execution times of the corresponding task on different machines. The average variation along the rows is referred to as the machine heterogeneity [2]. Similarly, the numbers along a column of the ETC matrix indicate the estimated expected execution times of the machine for different tasks. The average variation along the columns is referred to as the task heterogeneity [2]. One classification of heterogeneity is to divide it into high heterogeneity and low heterogeneity. Based on the above idea, four categories were proposed for the ETC matrix in [2]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo).

The ETC matrix can be further classified into two classes, consistent and inconsistent [2], which are orthogonal to the previous classifications. For a consistent ETC matrix, if machine m_x has a lower execution time than machine m_y for task t_k, then the same is true for any task t_i. The ETC matrices that are not consistent are inconsistent ETC matrices. Inconsistent ETC matrices occur in practice when: (1) there is a variety of different machine architectures in the HC suite (e.g., parallel machines, superscalars, workstations), and (2) there is a variety of different computational needs among the tasks (e.g., readily parallelizable tasks, difficult to parallelize tasks, tasks that are floating point intensive, simple text formatting tasks). Thus, the way in which a task's needs correspond to a machine's capabilities may differ for each possible pairing of tasks to machines.

As a subclass of inconsistent ETC matrices, a semi-consistent class could also be defined. A semi-consistent ETC matrix is characterized by a consistent sub-matrix. In the semi-consistent ETC matrices used here, 50% of the tasks and 25% of the machines define a consistent sub-matrix. Furthermore, it is assumed that for a particular task the execution times that fall within the consistent sub-matrix are smaller than those that fall out. This assumption is justified because one way for some machines to perform consistently for some tasks is to be very much faster for those tasks than the other machines.

Let an ETC matrix have t_{max} rows and m_{max} columns. Random ETC matrices that belong to the different categories are generated in the following manner:

1. Let $\underline{\Gamma_t}$ be an arbitrary constant quantifying task heterogeneity, being smaller for low task heterogeneity. Let $\underline{N_t}$ be a number picked from the uniform random distribution with range $[1, \Gamma_t]$.
2. Let $\underline{\Gamma_m}$ be an arbitrary constant quantifying machine heterogeneity, being smaller for low machine heterogeneity. Let $\underline{N_m}$ be a number picked from the uniform random distribution with range $[1, \Gamma_m]$.
3. Sample N_t t_{max} times to get a vector $\underline{q}[0..(t_{max}-1)]$.
4. Generate the ETC matrix, $\underline{e}[0..(t_{max}-1), 0..(m_{max}-1)]$ by the following algorithm:

```
for $t_i$ from 0 to $(t_{max} - 1)$
    for $m_j$ from 0 to $(m_{max} - 1)$
        pick a new value for $N_m$
        $e[i, j] = q[i] * N_m$
    endfor
endfor
```

From the raw ETC matrix generated above, a semi-consistent matrix could be generated by sorting the execution times across a random subset of the machines for each task in a random subset of tasks. An inconsistent ETC matrix could be obtained simply by leaving the raw ETC matrix as such. Consistent ETC matrices were not considered in this study because they are least likely to arise in the current intended MSHN environment.

In the experiments described here, the values of Γ_t for low and high task heterogeneities are 1000 and 3000, respectively. The values of Γ_m for low and high machine heterogeneities are 10 and 100, respectively. These heterogeneous ranges are based on one type of expected environment for MSHN.

5. Experimental Results and Discussion

5.1. Overview

The experimental evaluation of the heuristics is performed in three parts. In the first part, the immediate mode heuristics are compared using various metrics. The second part involves a comparison of the batch mode heuristics. The third part is the comparison of the batch mode and the immediate mode heuristics. Unless stated otherwise, the following are valid for the experiments described here. The number of machines is held constant at 20, and the experiments are performed for $| K | = \{1000, 2000\}$. All heuristics are evaluated in a HiHi heterogeneity environment, both for the inconsistent and the semi-consistent cases, because these correspond to some of the currently expected MSHN environments.

For each value of $| K |$, tasks are mapped under two different Poisson arrival rates, $\underline{\lambda_h}$ and $\underline{\lambda_l}$, such that $\lambda_h > \lambda_l$. The value of λ_h is chosen empirically to be high enough to allow at most 50% tasks to have completed when the last task in the set arrives. That is, for λ_h, when at least 50% of the tasks execute no new tasks are arriving. This may correspond to a situation when tasks are submitted during the day but not at night.

In contrast, λ_l is chosen to be low enough to allow at least 90% of the tasks to have completed when the last task in the set arrives. That is, for λ_l, when at most 10% of the tasks execute no new tasks are arriving. This may correspond more closely than λ_h to a situation where tasks arrive continuously. The difference between λ_h and λ_l can also be considered to represent a difference in burstiness.

Some experiments were also performed at a third arrival rate $\underline{\lambda_t}$, where λ_t was high enough to ensure that only 20% of the tasks have completed when the last task in the set arrived. The MCT heuristic was used as a basis for these percentages. Unless otherwise stated, the task arrival rate is set to λ_h.

Example comparisons are discussed in Subsections 5.2 to 5.4. Each data point in the comparison charts is an average over 50 trials, where for each trial the simulated actual task execution times are chosen independently. The makespan for each trial for each heuristic has been normalized with respect to the benchmark heuristic, which is the MCT for immediate mode heuristics, and the Min-min for the batch mode heuristics. The Min-min serves as a benchmark also for the experiments where batch mode heuristics are compared with immediate mode heuristics. Each bar (except the one for the benchmark heuristic) in the comparison charts gives a 95% confidence interval (shown as an "I" on the top of bars) for the mean of the normalized value. Occasionally the upper bound, lower bound, or the entire confidence interval is not distinguishable from the mean value for the scale used in the graphs here. More general conclusions about the heuristics' performance are in Section 6.

5.2. Comparisons of the immediate mode heuristics

Unless otherwise stated, the immediate mode heuristics are investigated under the following conditions. In the KPB heuristic, k is equal to 20%. This particular value of k was found to give the lowest makespan for the KPB heuristic under the conditions of the experiments. For the SA, the lower threshold and the upper threshold for the load balance index are 0.6 and 0.9, respectively. Once again these values were found to give optimum values of makespan for the SA.

In Figure 4, the immediate mode heuristics are compared based on normalized makespan for inconsistent HiHi heterogeneity. From Figure 4, it can be noted that the KPB heuristic completes the execution of the last finishing task earlier than the other heuristics (however, it is only slightly better than the MCT). For $k = 20\%$ and $m = 20$, the KPB heuristic forces a task to choose a machine from a subset of four machines. These four machines have the lowest execution times for the given task. The chosen machine would give the smallest completion time as compared to other machines in the set.

Figure 5 compares the normalized makespans of the different immediate mode heuristics for semi-consistent HiHi heterogeneity. As shown in Figures 4 and 5, the relative performance of the different immediate mode heuristics is impacted by the degree of consistency of the ETC matrices. However, the KPB still performs best, closely followed by the MCT.

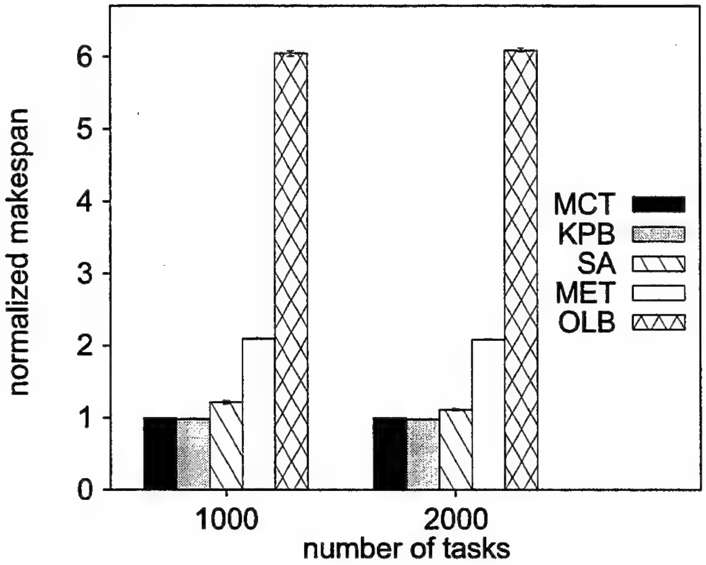


Figure 4: Makespan for the immediate mode heuristics for inconsistent HiHi heterogeneity.

For the semi-consistent type of heterogeneity, machines within a particular subset perform tasks that lie within a particular subset faster than other machines. From Figure 5, it can be observed that for semi-consistent ETC matrices, the MET heuristic performs the worst. For the semi-consistent matrices used in these simulations, the MET heuristic maps half of the tasks to the same machine, considerably increasing the load imbalance. Although the KPB considers only the fastest four machines for each task for the particular value of *k* used here (which happen to be the same four machines for half of the tasks), the performance does not differ much from the inconsistent HiHi case. Additional experiments have shown that the KPB performance is quite insensitive to values of k as long as k is larger than the minimum value (where the KPB heuristic is reduced to the MET heuristic). For example, when k is doubled from its minimum value of 5%, the makespan decreases by a factor of about five. However a further doubling of k brings down the makespan by a factor of only about 1.2.

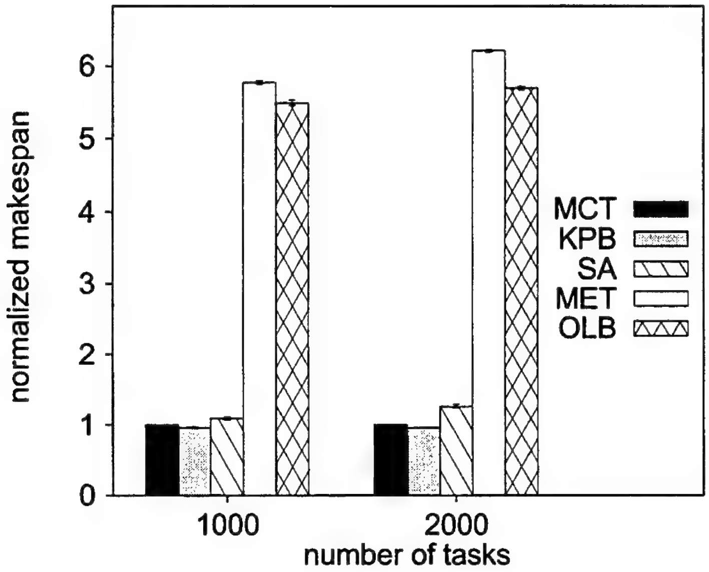


Figure 5: Makespan of the immediate mode heuristics for semi-consistent HiHi heterogeneity.

5.3. Comparisons of the batch mode heuristics

Figure 6 compares the batch mode heuristics based on normalized makespan. In these comparisons, unless otherwise stated, the regular time interval strategy is employed to schedule meta-task mapping events. The time interval is set to 10 seconds. This value was empirically found to optimize makespan over other values. From Figure 6, it can be noted that the Sufferage heuristic outperforms the Min-min and the Max-min heuristics based on makespan (although, it is only slightly better than the Min-min). The Sufferage heuristic considers the "loss" in completion time of a task if it is not assigned to its first choice in making

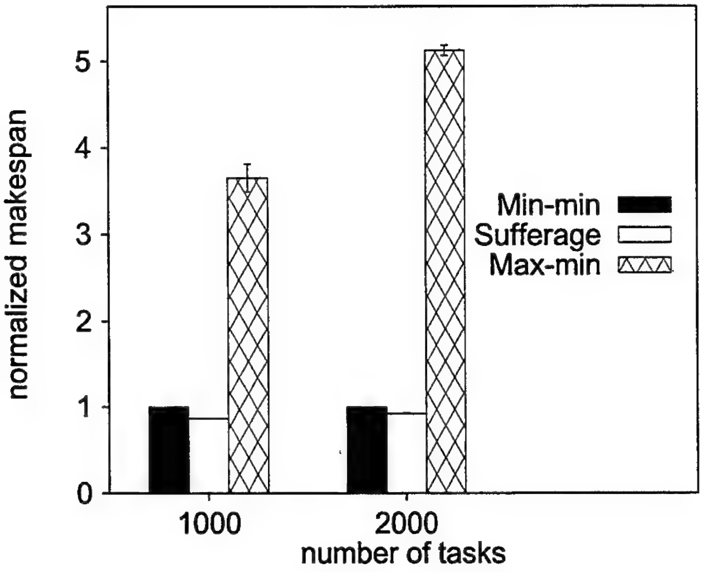


Figure 6: Makespan of the batch mode heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity.

the mapping decisions. By assigning their first choice machines to the tasks that have the highest sufferage values among all contending tasks, the Sufferage heuristic reduces the overall completion time.

Furthermore, it can be noted that the makespan given by the Max-min is much larger than the makespans obtained by the other two heuristics. Using reasoning similar to that given in Subsection 3.3 for explaining better expected performance for the Min-min, it can be seen that the Max-min assignments change a given machine's ready time status by a larger amount than the Min-min assignments do. If tasks t_i and t_k are contending for a particular machine m_j, then the Max-min assigns m_j to the task (say t_i) that will increase the ready time of m_j more. This decreases the probability that t_k will still have its earliest completion time on m_j and shall be assigned to it. Experimental data shows that the percentage of tasks assigned their minimum execution time machine is likely to be lower for the Max-min than for other batch mode heuristics [17]. It might be expected that a larger makespan will result if a larger number of tasks is assigned to the machines that do not have the best execution times for those tasks. Although not shown here, the results for makespan for semi-consistent HiHi are similar to those for inconsistent HiHi.

The impact of aging on batch mode heuristics is shown in Figure 7. The Min-min without aging is used here to normalize the performance of the other heuristics. The Max-min benefits most from the aging scheme. Recall that the Min-min performs much better than the Max-min when there is no aging. Aging modifies the Max-min's operation so that tasks with smaller completion times can be scheduled prior to those with larger completion times, thus reducing the negative aspects of that technique. This is discussed further in [17].

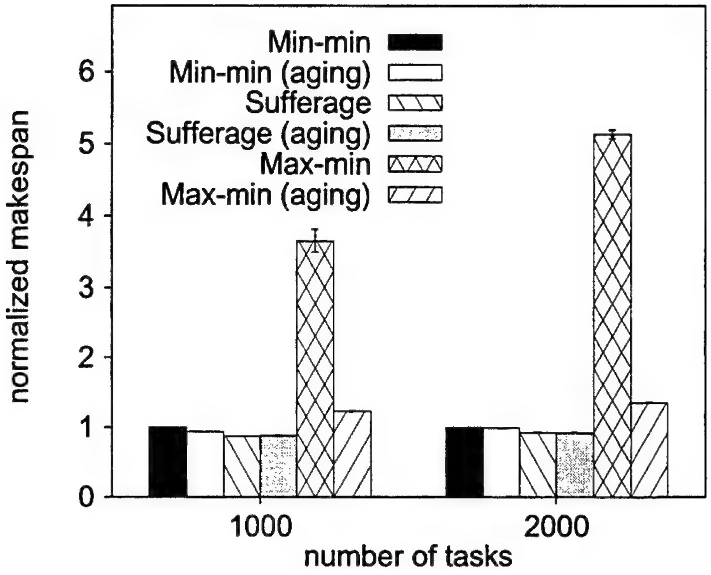


Figure 7: Makespan for the batch mode heuristics for the regular time interval strategy with and without aging for inconsistent HiHi heterogeneity.

Figure 8 shows the result of repeating the above experiments with a fixed count strategy for a batch size of 40. This particular batch size was found to give an optimum value of the makespan for the Min-min heuristic. The Min-min with regular time interval strategy (interval of ten seconds) is used here to normalize the performance of the other heuristics. Figure 8 compares regular time interval strategy and fixed count strategy on the basis of normalized makespans given by different heuristics for inconsistent HiHi heterogeneity. It can be seen that the fixed count approach gives similar results for the Min-min and the Sufferage heuristics. The Max-min heuristic, however, benefits considerably from the fixed count approach; makespan drops to about 60% for $|K| = 1000$, and to about 50% for $|K| = 2000$ as compared to the makespan given by the regular time interval strategy. A possible explanation lies in a conceptual element of similarity between the fixed count approach and the aging scheme. The value of $\kappa = 40$ used here resulted in batch sizes that were smaller than those using the ten second regular time interval strategy. Thus, small tasks waiting to execute will have fewer tasks to compete with, and, hence, less chance of being delayed by a larger task. Although not shown here, the results for the semi-consistent case show that as compared to the inconsistent case, the regular time interval approach gives slightly better results than the fixed count approach for the Sufferage and the Min-min. For the Max-min, however, for both inconsistent and semi-consistent cases, the fixed count strategy gives a much larger improvement over the regular time strategy.

It should be noted that all the results given here are for inconsistent HiHi heterogeneity. For other types of heterogeneity the results might be different. For example, for inconsistent LoLo heterogeneity, the performance of the Max-min is almost identical to that of the Min-min [17].

5.4. Comparing immediate mode and batch mode heuristics

In Figure 9, two immediate mode heuristics, the MCT and the KPB, are compared with two batch mode heuristics, the Min-min and the Sufferage. The comparison is performed with Poisson arrival rate set to λ_h. It can be noted that for this "high" arrival rate and $|K| = 2000$, batch mode heuristics are superior to

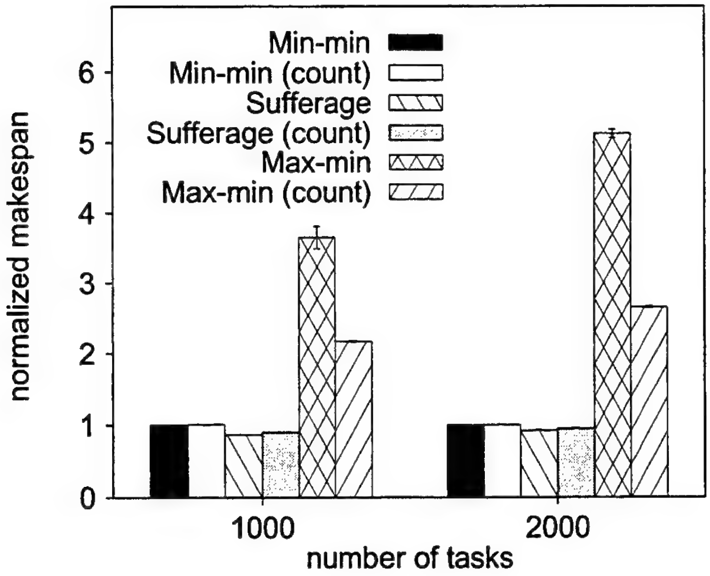


Figure 8: Comparison of the makespans given by the regular time interval strategy and the fixed count strategy for inconsistent HiHi heterogeneity.

immediate mode heuristics. This is because the number of tasks waiting to begin execution is likely to be larger in the above circumstances than in any other considered here, which in turn means that rescheduling is likely to improve many more mappings in such a system. The immediate mode heuristics consider only one task when they try to optimize machine assignment, and do not reschedule. Recall that the mapping heuristics use a combination of expected and actual task execution times to compute machine ready times. The immediate mode heuristics are likely to approach the performance of the batch ones at low task arrival rates, because then both classes of heuristics have comparable information about the actual execution times of the tasks. For example, at a certain low arrival rate, the 100-th arriving task might find that 70 previously arrived tasks have completed. At a higher arrival rate, only 20 tasks might have completed when the 100-th task arrived. The above observation is supported by the graph in Figure 10, which shows that the relative performance difference between immediate and batch mode heuristics decreases with a decrease in arrival rate. Given the observation that the KPB and the Sufferage perform almost similarly at this low arrival rate, it might be better to use the KPB heuristic because of its smaller computational complexity.

Figure 11 shows the performance difference between immediate and batch mode heuristics at an even faster arrival rate of λ_f. It can be seen that for $|K| = 2000$ batch mode heuristics outperform immediate mode heuristics with a larger margin here. Although not shown in the results here, the makespan values for all heuristics are larger for lower arrival rate. This is attributable to the fact that at lower arrival rates, there typically is more machine idle time.

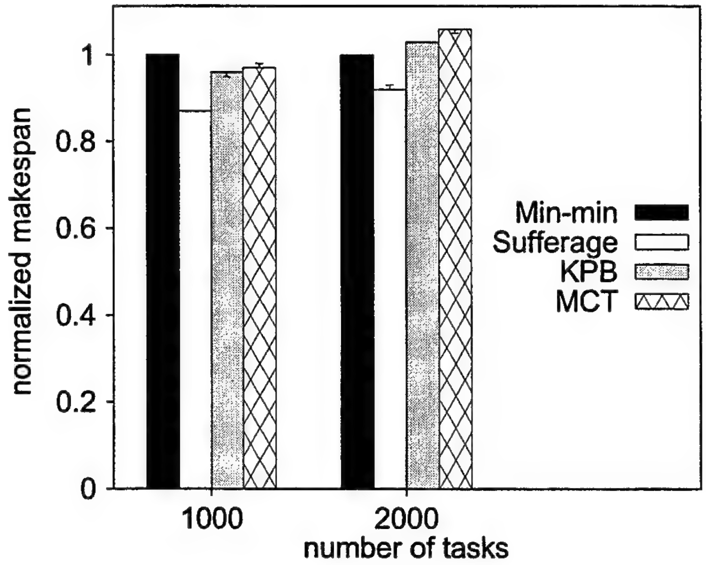


Figure 9: Comparison of the makespan given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_h.

6. Conclusions

New and previously proposed dynamic matching and scheduling heuristics for mapping independent tasks onto HC systems were compared under a variety of simulated computational environments. Five

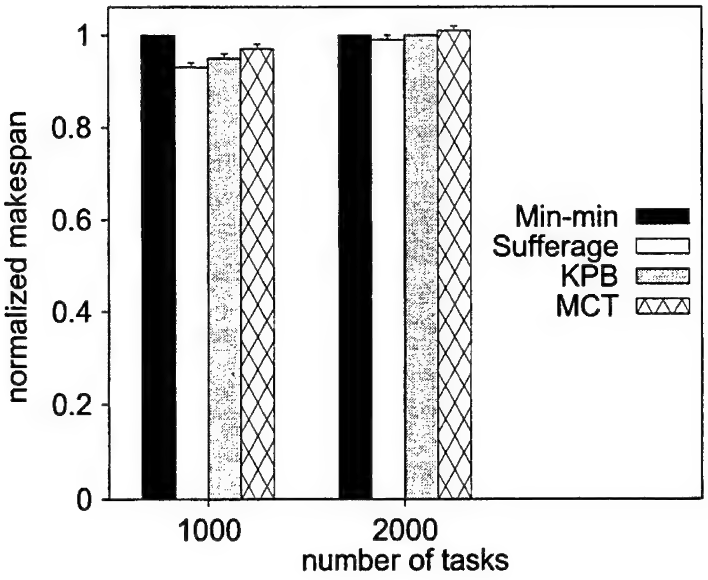


Figure 10: Comparison of the makespan given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_l.

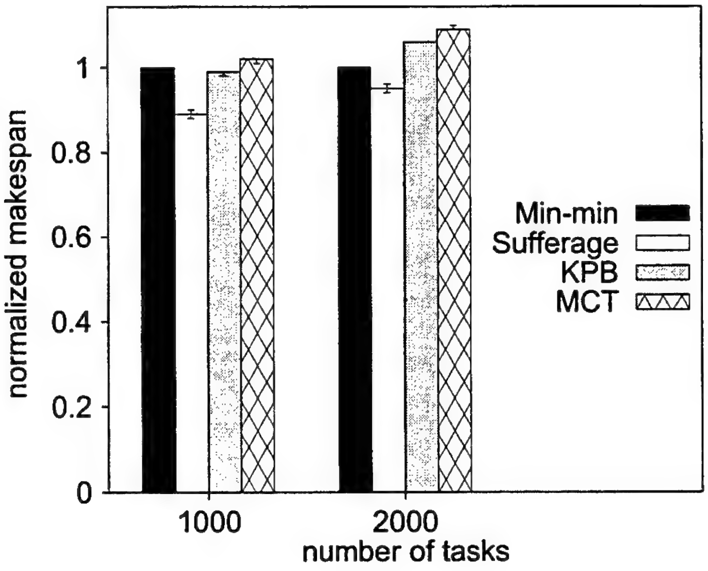


Figure 11: Comparison of the makespan given by batch mode heuristics (regular time interval strategy) and immediate mode heuristics for inconsistent HiHi heterogeneity and an arrival rate of λ_t.

immediate mode heuristics and three batch mode heuristics were studied.

In the immediate mode, for both the semi-consistent and the inconsistent types of HiHi heterogeneity, the KPB heuristic outperformed the other heuristics (however, the KPB was only slightly better than the MCT). The relative performance of the OLB and the MET with respect to the makespan reversed when the heterogeneity was changed from the inconsistent to the semi-consistent. The OLB did better than the MET for the semi-consistent case.

In the batch mode, for the semi-consistent and the inconsistent types of HiHi heterogeneity, the Sufferage performed the best (though, the Sufferage was only slightly better than the Min-min). The batch mode heuristics were shown to give a smaller makespan than the immediate ones for large $| K |$ and high task arrival rate. For smaller values of $| K |$ and lower task arrival rates, the improvement in makespan offered by batch mode heuristics was shown to be nominal.

This study quantifies how the relative performance of these dynamic mapping heuristics depends on (a) the consistency property of the ETC matrix, and (b) the arrival rate of the tasks. Thus, the choice of the heuristic that is best to use in a given heterogeneous environment will be a function of such factors. Therefore, it is important to include a set of heuristics in a resource management system for HC, and then use the heuristic that is most appropriate for a given situation (as will be done in the Scheduling Advisor for MSHN).

Researchers can build on the evaluation techniques and results presented here in future efforts by considering other non-preemptive dynamic heuristics, as well as preemptive ones. Furthermore, in future studies, tasks can be characterized in more complex ways (e.g., inter-task communications, deadlines, priorities [3]) and using other environmental factors (e.g., task arrival rates, degrees of heterogeneity, number of machines in the HC suite, impact of changing the variance when simulating actual task execution times). Thus, the studies given in this paper illustrate some evaluation techniques, examine important heuristics, and provide comparisons, as well as act as a framework for future research.

Acknowledgments: A preliminary version of portions of this paper was presented at the *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, April 1999. The authors thank Taylor Kidd, Surjamukhi Chatterjea, and Tracy D. Braun for their valuable comments and suggestions.

References

[1] R. Armstrong, D. Hensgen, and T. Kidd, The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions, *in* "7th IEEE Heterogeneous Computing Workshop (HCW '98)," pp. 79–87, 1998.

[2] R. Armstrong, "Investigation of Effect of Different Run-Time Distributions on SmartNet Performance," Master's thesis, Department of Computer Science, Naval Postgraduate School, 1997 (D. Hensgen, Advisor).

[3] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems, *in* "1998 IEEE Symposium on Reliable Distributed Systems," pp. 330–335, 1998.

[4] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, R. F. Freund, and D. Hensgen, A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, *in* "8th IEEE Heterogeneous Computing Workshop (HCW '99)," pp. 15-29, 1999.

[5] A. H. Buss, A tutorial on discrete-event modeling with simulation graphs, *in* "1995 Winter Simulation Conference (WSC '95)," pp. 74–81, 1995.

[6] M. M. Eshaghian (ed.), "Heterogeneous Computing," Artech House, Norwood, MA, 1996.

[7] I. Foster and C. Kesselman (eds.), "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, San Fransisco, CA, 1999.

[8] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, Scheduling resources in multiuser, heterogeneous, computing environments with SmartNet, *in* "7th IEEE Heterogeneous Computing Workshop (HCW '98)," pp. 184–199, 1998.

[9] R. F. Freund and H. J. Siegel, Heterogeneous processing, *IEEE Computer*, **26**, No. 6 (June 1993), 13–17.

[10] A. Ghafoor and J. Yang, Distributed heterogeneous supercomputing management system, *IEEE Computer*, **26**, No. 6 (June 1993), 78–86.

[11] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, An overview of MSHN: The Management System for Heterogeneous Networks, *in* "8th IEEE Heterogeneous Computing Workshop (HCW '99)," pp. 184–198, 1999.

[12] O. H. Ibarra and C. E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, *Journal of the ACM*, **24**, No. 2 (Apr. 1977), 280–289.

[13] M. A. Iverson and F. Ozguner, Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment, *in* "7th IEEE Heterogeneous Computing Workshop (HCW '98)," pp. 70–78, 1998.

[14] R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, Inc., New York, NY, 1991.

[15] M. Kafil and I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, *IEEE Concurrency*, **6**, No. 3 (July-Sep. 1998), 42–51.

[16] C. Leangsuksun, J. Potter, and S. Scott, Dynamic task mapping algorithms for a distributed heterogeneous computing environment, *in* “4th IEEE Heterogeneous Computing Workshop (HCW ’95),” pp. 30–34, 1995.

[17] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund, “A Comparison of Dynamic Strategies for Mapping a Class of Independent Tasks onto Heterogeneous Computing Systems,”, Technical Report, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 1999, in preparation.

[18] M. Maheswaran, T. D. Braun, and H. J. Siegel, Heterogeneous distributed computing, *in* “Encyclopedia of Electrical and Electronics Engineering” (J. G. Webster, Ed.), Vol. 8, pp. 679–690, John Wiley, New York, NY, 1999.

[19] R. Mirchandaney, D. Towsley, and J. A. Stankovic, Adaptive load sharing in heterogeneous distributed systems, *Journal of Parallel and Distributed Computing*, **9**, No. 4 (Aug. 1990), 331–346.

[20] A. Papoulis, “Probability, Random Variables, and Stochastic Processes,” McGraw-Hill, New York, NY, 1984.

[21] M. Pinedo, “Scheduling: Theory, Algorithms, and Systems,” Prentice Hall, Englewood Cliffs, NJ, 1995.

[22] U. W. Pooch and J. A. Wall, “Discrete Event Simulation: A Practical Approach,” CRC Press, Boca Raton, FL, 1993.

[23] H. G. Rotithor, Taxonomy of dynamic task scheduling schemes in distributed computing systems, *IEE Proceedings on Computer and Digital Techniques*, **141**, No. 1 (Jan. 1994), 1–10.

[24] H. Singh and A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms, *in* "5th IEEE Heterogeneous Computing Workshop (HCW '96)," pp. 86–97, 1996.

[25] V. Suresh and D. Chaudhuri, Dynamic rescheduling–A survey of research, *International Journal of Production Economics*, **32**, No. 1 (Aug. 1993), 53–63.

[26] P. Tang, P. C. Yew, and C. Zhu, Impact of self-scheduling on performance of multiprocessor systems, *in* "3rd International Conference on Supercomputing," pp. 593–603, 1988.

[27] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, *Journal of Parallel and Distributed Computing*, **47**, No. 1 (Nov. 1997), 8–22.

Biographies

Muthucumaru Maheswaran is an Assistant Professor in the Department of Computer Science at the University of Manitoba, Canada. In 1990, he received a BSc degree in electrical and electronic engineering from the University of Peradeniya, Sri Lanka. He received an MSEE degree in 1994 and a PhD degree in 1998, both from the School of Electrical and Computer Engineering at Purdue University. He held a Fulbright scholarship during his tenure as an MSEE student at Purdue University. His research interests include computer architecture, distributed computing, heterogeneous computing, Internet and world wide web systems, metacomputing, mobile programs, network computing, parallel computing, resource management systems for metacomputing, and scientific computing. He has authored or coauthored 15 technical papers in these and related areas. He is a member of the Eta Kappa Nu honorary society.

Shoukat Ali is pursuing an MSEE degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a Research Assistant. His main research topic is dynamic mapping of meta-tasks in heterogeneous computing systems. He has held teaching positions at Aitchison College and Keynesian Institute of Management and Sciences, both in Lahore, Pakistan. He was also a Teaching Assistant at Purdue. Shoukat received his BS degree in electrical and electronic engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1996. His research interests include computer architecture, parallel computing, and heterogeneous computing.

Howard Jay Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received BS degrees in both electrical engineering and management from MIT, and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale*

Parallel Processing. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Debra Hensgen is an Associate Professor in the Computer Science Department at The Naval Postgraduate School. She received her PhD in the area of Distributed Operating Systems from the University of Kentucky. She is currently a Principal Investigator of the DARPA-sponsored Management System for Heterogeneous Networks QUORUM project (MSHN) and a co-investigator of the DARPA-sponsored Server and Active Agent Management (SAAM) Next Generation Internet project. Her areas of interest include active modeling in resource management systems, network re-routing to preserve quality of service guarantees, visualization tools for performance debugging of parallel and distributed systems, and methods for aggregating sensor information. She has published numerous papers concerning her contributions to the Concurra toolkit for automatically generating safe, efficient concurrent code, the Graze parallel processing performance debugger, the SAAM path information base, and the SmartNet and MSHN Resource Management Systems.

Richard F. Freund is a founder and CEO of NOEMIX, a San Diego based startup to commercialize distributed computing technology. Freund is also one of the early pioneers in the field of distributed computing, in which he has written or co-authored a number of papers. In addition he is a founder of the Heterogeneous Computing Workshop, held each year in conjunction with the International Parallel and Distributed Processing Symposium. Freund won a Meritorious Civilian Service Award during his former career as a government scientist.

From the proceedings of the *6th Euromicro Workshop on Parallel and Distributed Processing*,
Madrid, Spain, Jan. 1998, pp. 3–9. Invited keynote presentation.

High-Performance Mixed-Machine Heterogeneous Computing

Muthucumaru Maheswaran, Tracy D. Braun, and Howard Jay Siegel
Parallel Processing Laboratory, School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285 USA

Abstract

The focus of this invited keynote paper (to be presented by H. J. Siegel) is mixed-machine heterogeneous computing, where a suite of different kinds of high-performance machines are interconnected by high-speed links. Such a system can be orchestrated to perform an application whose subtasks have diverse execution requirements. Subtasks are assigned to and executed on the machines that will result in a minimal overall execution time for the task, considering factors including inter-machine communication overhead. A conceptual model of the automatic decomposition of tasks and assignment of subtasks is presented. Examples of static and dynamic approaches to the matching and scheduling of subtasks are summarized. Some open research problems are discussed.

1 Introduction

Existing high-performance computers typically achieve only a fraction of their peak capabilities on certain portions of some application tasks [8]. This is because different subtasks of an application can have very different computational requirements that result in the need for different machine capabilities. A single machine architecture cannot satisfy all the computational requirements of certain applications equally well. Thus, the use of a heterogeneous computing environment is more appropriate.

The focus of this invited keynote paper (to be presented by H. J. Siegel) is mixed-machine heterogeneous computing (HC), where a suite of different kinds of high-performance machines are interconnected by high-speed links. Such a system provides a variety of architectural capabilities, orchestrated to perform an application whose subtasks have diverse execution requirements [17]. The task must be decomposed into subtasks, where each subtask is computationally homogeneous, and different subtasks may have different machine architectural requirements. These subtasks may share initial or generated data, creating the potential for inter-machine data transfer overhead. To exploit HC systems in such situations, each subtask must be assigned to and executed on the machines that will result in a minimal overall execution time for the task, considering factors including inter-machine communication overhead. The subtasks can be assigned either prior to execution (statically) or during the execution (dynamically).

Figure 1 shows a hypothetical example of an application program whose various subtasks are best suited for execution on different machine architectures [8]. Executing the whole program on a SIMD machine only gives approximately five times the performance achieved by a baseline serial machine. Only the SIMD portion of the program can be executed significantly faster because of the mismatch between each subtask's unique computational requirement and the SIMD architecture. Alternatively, the use of four different machines, each matched to the computational requirements of the subtask to which it was assigned, can result in an execution 50 times as fast as the baseline serial machine.

A conceptual model for automatic HC is introduced in Section 2. As an example of current research in static matching and scheduling, Section 3 presents a genetic-algorithm-based approach. Section 4 describes a dynamic mapping algorithm for on-line matching and scheduling. Open problems are discussed in Section 5.

2 A Conceptual Model for HC

Typically, users of HC systems must perform task decomposition and subtask matching and scheduling themselves (e.g.,[13, 15, 20]). A conceptual model for automatic task decomposition, matching subtasks to machines, and scheduling subtasks is shown in Figure 2. It builds on the model presented in [18] and is referred to as a "conceptual" model because no complete automatic implementation currently exists.

In stage 1, using information about the expected types of application tasks and about the machines in the HC suite, a set of parameters is generated that

This research is supported by the DARPA/ITO Quorum Program under NPS subcontract number N62271-97-M-0900.

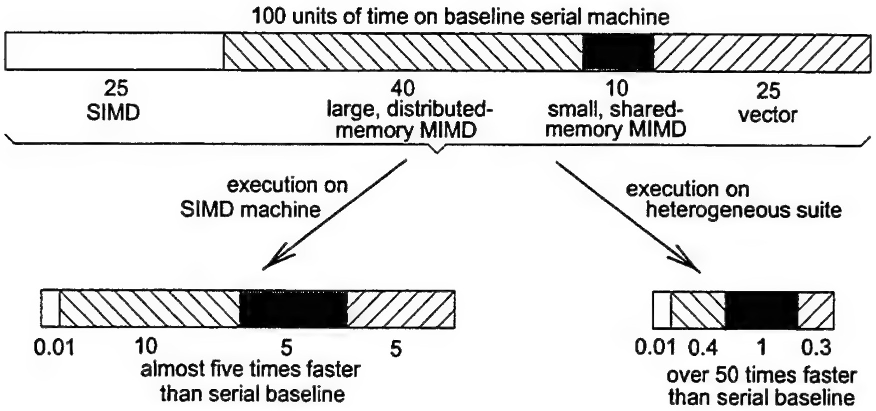

100 units of time on baseline serial machine
25
SIMD
40
large, distributed-memory MIMD
10
small, shared-memory MIMD
25
vector
execution on SIMD machine
execution on heterogeneous suite
0.01
10
5
5
almost five times faster than serial baseline
0.01
0.4
1
0.3
over 50 times faster than serial baseline

Figure 1: Hypothetical example of the advantage of using a heterogeneous suite of machines, where the heterogeneous suite time includes intermachine communication overhead (based on [8]). Not drawn to scale.

is relevant to both the computational requirements of the applications and the machine capabilities of the HC system. For each parameter, categories for computational characteristics and categories for machine architecture features are derived.

In stage 2, task profiling decomposes the application task into subtasks, each of which is computationally homogeneous. The computational requirements for each subtask are then quantified by profiling the code and data. Analytical benchmarking is used to quantify how effectively each of the available machines in the suite performs on each of the types of computations being considered.

One of the functions of stage 3 is to be able to use the information from stage 2 to derive the estimated execution time of each subtask on each machine in the HC suite and the associated inter-machine communication overhead. Then, these static results, with the machine and inter-machine network initial loading and "status" (i.e., machine/network faults and expected subtask/transfer completion times) are used to generate an assignment of the subtasks to machines and an execution schedule based on certain cost metrics (e.g., minimizing the overall task execution time).

Stage 4 is the execution of the given application. In dynamic matching and scheduling systems, the subtask completion times and loading/status of the machines/network are monitored. This information may be used to reinvoke the matching and scheduling of stage 3 to improve the machine assignment and execution schedule.

Automatic HC is relatively new field. Frameworks for task profiling, analytical benchmarking, and mapping (matching and scheduling) have been proposed, however, further research is needed to make this conceptual model a reality [17, 18].

3 Static Task Mapping

In general, the problem of performing matching and scheduling in an HC environment is NP-complete [5], and therefore some heuristic must be employed. A variety of static approaches have been studied for different HC models (e.g., [4, 11, 16, 19]). As an example of current HC research on mapping statically, a genetic-algorithm approach from [21] is summarized. An application task is decomposed into a set of subtasks $\underline{S}$. Let s_i be the i-th subtask. An HC suite consists of a set of machines $\underline{M}$. Let m_j be the j-th machine. Each machine can be a different type. The global data items are data items that need to be transferred between subtasks.

The following assumptions about the applications and HC environment are made. Each application task will be represented by a DAG (directed acyclic graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies between subtasks. (Note that while the subtasks' dependencies are represented as a DAG, subtasks themselves may contain loops.) Each edge is labeled by the global data item that is transferred over it. The application task has exclusive use of the HC environment, and the genetic-algorithm-based mapper controls the HC machine suite (hardware platform). Subtask execution is non-preemptive. The estimated expected execution time of each subtask on each machine is known. For each pair of machines in the HC suite, an equation for estimating the time to send data between those machines as a function of data set size is known.

Genetic algorithms (GAs) are a heuristic approach

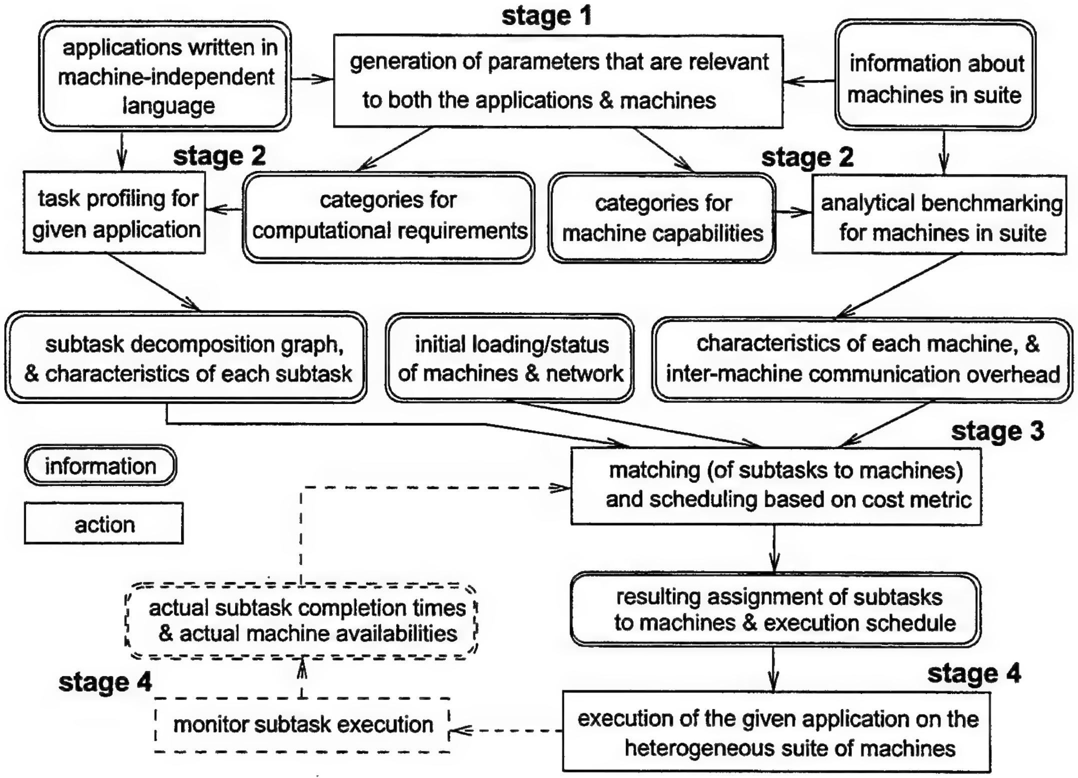

stage 1
applications written in machine-independent language
generation of parameters that are relevant to both the applications & machines
information about machines in suite
stage 2
stage 2
task profiling for given application
categories for computational requirements
categories for machine capabilities
analytical benchmarking for machines in suite
subtask decomposition graph, & characteristics of each subtask
initial loading/status of machines & network
characteristics of each machine, & inter-machine communication overhead
stage 3
information
action
matching (of subtasks to machines) and scheduling based on cost metric
actual subtask completion times & actual machine availabilities
resulting assignment of subtasks to machines & execution schedule
stage 4
stage 4
monitor subtask execution
execution of the given application on the heterogeneous suite of machines

Figure 2: Model for integrating the software support needed for automating the use of heterogeneous computing systems (based on [18]).

to optimization problems that are intractable. The first step is to encode some of the possible solutions as chromosomes, the set of which is referred to as a population. In the [21] approach, each chromosome consists of two parts: the matching string (mat) and the scheduling string (ss). If mat(i) = j, then subtask s_i is assigned to machine m_j. Typically, multiple subtasks will be assigned to the same machine, and then executed in a non-preemptive manner based on an ordering that obeys the precedence constraints (data dependencies) specified in the task DAG. The scheduling string is a topological sort of the DAG representing the task (i.e., a valid total ordering of the partially ordered DAG). If ss(k) = i, then subtask s_i is the k-th subtask in the total ordering. Because it is a topological sort, if subtask ss(k) needs a global data item created by subtask ss(j), then $j < k$. The scheduling string gives an ordering of subtasks that is used by the evaluation step.

Each chromosome is associated with a fitness value, which is the completion time of the solution (i.e., mapping) represented by this chromosome (i.e., the expected execution time of the application task if the mapping specified by this chromosome were used). Overlapping among all of the computations and communications performed is limited only by inter-subtask data dependencies and the availability of the machines and the inter-machine network.

In the initial population generation step, a predefined number of distinct chromosomes are randomly created. The solution from a non-evolutionary heuristic is also included in the initial population. After the initial population is determined, the genetic algorithm iterates until a predefined stopping criterion is met. Each iteration consists of the selection, crossover, mutation, and evaluation steps.

In the selection step, some members of the population are removed and others are duplicated. First, all of the chromosomes in the population are ordered (ranked) by their fitness values. Then a rank-based roulette wheel selection scheme is used to implement proportionate selection. The population size is kept constant and a chromosome representing a better solution has a higher probability of having one or more copies in the next generation population. This GA approach also incorporates elitism, i.e., the best solution found so far is always kept in the population.

The selection step is followed by the crossover step, where some chromosomes are paired and corresponding components of the paired chromosomes are exchanged. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings in the current population. For each pair, it randomly generates a cutoff point, and divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered using the relative positions of these subtasks in the other original scheduling string, thus guaranteeing that the newly generated scheduling strings are valid schedules. The crossover operator for the matching strings randomly chooses some pairs of the matching strings in the current population. For each pair, it randomly generates a cutoff point, and divides both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The next step is mutation. The scheduling string mutation operator randomly chooses some scheduling strings in the current population. Then for each chosen scheduling string, it randomly selects a victim subtask. The valid range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed and still have a valid topological sort of the DAG. The victim subtask is moved randomly to another position in the scheduling string within its valid range. The matching string mutation operator randomly chooses some matching strings in the current population. For each chosen matching string, it randomly selects a subtask entry. Then the machine assignment for the selected entry is changed randomly to another machine.

The last step of an evolution iteration is the evaluation step to determine the fitness value of each chromosome in the current population. A communication subsystem that is modeled after a HiPPI LAN with a central crossbar switch was assumed for the tests that were conducted. As stated earlier, the above steps of selection, crossover, mutation, and evaluation are repeated until one of the stopping criteria are met: (1) the number of iterations reaches some limit (e.g., 1000), (2) the population converged (all the chromosomes had the same fitness value), or (3) the best solution found was not improved after some number of iterations (e.g., 150).

In the tests of this GA approach in [21], simulated program behaviors were used. Small-scale tests were conducted with up to ten subtasks, three machines, and population size 50. For each test, the GA approach found a solution (mapping) that had the same expected completion time for the task as that of the optimal solution found by exhaustive search. Larger tests with up to 100 subtasks, 20 machines, and population size 200 were conducted. This GA approach produced solutions (mappings) that averaged from 150% to 200% better than those produced by the non-evolutionary levelized min-time (LMT) heuristic proposed in [10]. The heuristic in [10] was selected for comparison because it used a similar model of HC. The GA approach took much more time to generate the mappings than did the LMT approach; however, if the mappings are being created off-line, prior to run time, for production tasks that will be executed repeatedly, the generation time is worthwhile.

4 Dynamic Task Mapping

The static mapping algorithms assume that accurate estimates are available for parameters such as subtask completion times. However, in general, such estimates have a degree of uncertainty in them. Therefore, dynamic mapping algorithms that can handle the uncertainty may be needed. Researchers have proposed different dynamic algorithms for varying HC models (e.g., [3, 6, 9, 14]). In dynamic mapping algorithms, machines can come on-line and go off-line at run time.

As an example of current Purdue research on dynamic mapping, an algorithm called the hybrid mapper is discussed in this section. The hybrid mapper uses some results based on an initial static mapping in conjunction with information available only at execution time. It is based on the list scheduling class of algorithms [10]. An initial, statically obtained mapping is provided as input to the hybrid mapper. If the initial mapping is not provided, it should be obtained before running the hybrid mapper.

An HC model similar to the one described in Section 3 is assumed here. Two indices, i and k are associated with each subtask $s_{i,k}$. The index i denotes that $s_{i,k}$ is the i-th subtask in the set S and k is the block number assigned to it by the partitioning algorithm described below. The estimated expected computation time of subtask $s_{i,k}$ on machine m_j is given by $e_{i,k,j}$. The earliest time at which machine m_j is available is given by $A[j]$.

The hybrid mapper executes in two phases. This first phase uses the initial static mapping, expected subtask computation times, and expected data transfer times to preprocess the DAG that represents the application off line. Initially, the DAG is partitioned into B blocks, numbered consecutively from 0 to $B-1$. The partitioning is done such that the subtasks within a block are independent, i.e., there are no data dependencies among the subtasks in a block. Furthermore, for each block k subtask, $s_{j,k}$, there exists at least one incident edge (data dependency) such that the source subtask is in block $k-1$, i.e., an incident edge from some $s_{i,k-1}$. The $(B-1)$-th block includes the subtasks without any successors and the 0-th block includes only those subtasks without any predecessors.

Once the subtasks in the DAG are partitioned, each subtask is assigned a level by examining the subtasks from block $B-1$ to block 0. The level of each subtask in the $(B-1)$-th block is set to its expected computation time on the machine to which it was assigned by the initial matching. Now consider the k-th block, $0 \leq k < B-1$. Recall $e_{i,k,x}$ is the expected computation time of the subtask $s_{i,k}$ on machine m_x. Let $c_{i,j}$ be the data transfer time for a descendent $s_{j,q}$ of $s_{i,k}$ to get all the relevant data items from $s_{i,k}$ (where $q \geq k+1$). The value of $c_{i,j}$ will be dependent on the machines assigned to subtasks $s_{i,k}$ and $s_{j,q}$ by the initial mapping. Let $\text{level}(s_{i,k})$ be the level of the subtask $s_{i,k}$. Let $\text{iss}(s_{i,k})$ be the immediate successor set of subtask $s_{i,k}$ such that there is an arc from $s_{i,k}$ to each member of $\text{iss}(s_{i,k})$ in the DAG. With these definitions, the level of a subtask $s_{i,k}$ that is initially assigned to m_x is given by:

$$\text{level}(s_{i,k}) = e_{i,k,x} + \max_{s_{j,q} \in \text{iss}(s_{i,k})} (c_{i,j} + \text{level}(s_{j,q})).$$

The level of a subtask can be interpreted as the length of the critical path from the point the given subtask is located on the DAG to a subtask with no successors. The hybrid mapper is based on the heuristic idea that by executing the subtasks with higher levels as quickly as possible, the overall expected completion time for the application can be minimized.

The second phase of the hybrid mapper involves the actual execution of the subtasks. The execution of the subtasks proceeds from block 0 to block $B-1$. A block k is considered to be executing if at least one subtask from block k is executing. The execution of several blocks can overlap with each other in time, i.e., subtasks from different blocks could be executing at the same time.

The hybrid mapper starts remapping the block k subtasks when the first block $(k-1)$ subtask begins its execution. When block k is being scheduled, it is highly likely that actual execution time information can be used for many subtasks from blocks 0 to $k-2$. There may be some subtasks from blocks 0 to $k-2$ that could still be executing or awaiting execution when subtasks from block k are being considered for remapping. For such subtasks, expected execution times are used.

In a list-scheduling type of algorithm, the subtasks are first ordered based on some priority. Then, each subtask is mapped by examining the list of subtasks from the highest priority subtask to the lowest priority subtask. The machine to which each subtask is assigned depends on the matching criterion used by the particular algorithm. In the hybrid mapper, the priority of a subtask is equal to the level of that subtask that was computed statically in the first phase. The matching criterion used for subtask $s_{i,k}$ is the minimization of the partial completion time, defined below.

Let m_x be the machine on which $s_{i,k}$ is being considered for execution. Then let $\text{pct}(s_{i,j}, x)$ denote the partial completion time of the subtask $s_{i,k}$ on machine m_x, $\text{dr}(s_{i,k})$ be the time at which the last data item required by $s_{i,k}$ to begin its execution arrives at m_x, and $\text{ips}(s_{i,k})$ be the immediate predecessor set for subtask $s_{i,k}$ such that there is an arc to $s_{i,k}$ from each member of $\text{ips}(s_{i,k})$ in the DAG. For any subtask $s_{i,k}$, where $s_{j,q} \in \text{ips}(s_{i,k})$, and $s_{j,q}$ is currently mapped onto machine m_y,

$$\begin{aligned} \text{dr}(s_{i,k}) &= \max_{s_{j,q} \in \text{ips}(s_{i,k})} (c_{j,i} + \text{pct}(s_{j,q}, y)), \\ \text{pct}(s_{i,k}, x) &= e_{i,k,x} + \max(A[x], \text{dr}(s_{i,k})). \end{aligned}$$

The subtask $s_{i,k}$ is remapped onto the machine m_x that gives the minimum $\text{pct}(s_{i,k}, x)$, and $A[x]$ is updated using $\text{pct}(s_{i,k})$. Then the next subtask from the list is considered for mapping.

The simulation results indicate that the performance of a statically obtained initial mapping can be improved by the hybrid mapper. Initial mappings were generated using the baseline algorithm [21]. The baseline algorithm partitions the subtasks into blocks using an algorithm similar to the one described here. Once the subtasks are partitioned into blocks, they are ordered such that a subtask in block k comes before a subtask in block l, where $k < l$. The subtasks within a block are arranged such that the subtasks with more descendents appear before the subtasks with less descendents (ties are broken arbitrarily). The subtasks are considered for assignment by traversing the list, beginning with block 0 subtasks. A subtask is mapped to the machine that gives the shortest partial completion time for the subtasks that have been mapped (including this subtask).

From the simulation results obtained, performance improvement from using the hybrid mapper can be as much as 15% for some cases. The timings also indicate that the remapping time needed per block of subtasks is in the order hundreds of milliseconds for up to 50 machines and 500 subtasks. In the worst case situation, to obtain complete overlap between the execution of the subtasks and the operation of the hybrid mapper, the computation time for the shortest running subtask must be greater than the per block remapping time. Ongoing research will examine ways to increase the performance gain obtained from the use of the hybrid remapper.

5 Open Research Problems

There are a great many open problems that need to be solved before HC can be made available to application programmers in a transparent way (summarized here from [12, 17, 18]). Implementation of the automatic HC programming environment envisioned in the conceptual model in Section 2 will require a great deal of research for devising practical and theoretically sound methodologies for each component of every stage. A general question that is particularly applicable to stages 1 and 2 of the conceptual model is: "What information should (must) the user provide and what information should (can) be determined automatically?"

To program an HC system, it would be best to have one or more machine-independent programming languages [22] that allow the user to augment the code with compiler directives. The language and directives should be designed to facilitate (a) the compilation of the program into efficient code for the machines in the suite, (b) the task decomposition, (c) the determination of computational requirements of each subtask, and (d) the use of machine-dependent subroutine libraries.

There is a need for debugging and performance tuning tools that can be used across an HC suite of machines. This involves research in the areas of distributed programming environments and visualization techniques.

Ideally, information about the current loading and status of the machines in the HC suite and the network should be incorporated into the mapping decisions. Methods must be developed for measuring the current loading, determining the status (e.g., faulty or not), and estimating the subtask completion times. Also, the uncertainty present in the estimated parameter values such as subtask completion times should be taken into consideration in determining the machine assignment and execution schedule.

There is much ongoing research in the area of inter-machine data transport. This research includes the hardware support required, the software protocols required, designing the network topology, computing the minimum-time path between two machines, and devising rerouting schemes in case of faults or heavy loads. Related to this is the data formatting problem, involving issues such as data type storage formats and sizes, byte ordering within data types, and machines' network-interface buffer sizes.

Another area of research is dynamic task migration between different parallel machines at execution time. Current research in this area involves determining how to move an executing task between different machines [1, 2] and how to use dynamic task migration for load rebalancing or fault tolerance.

Some of the future research outlined here may be pursued as part of a DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort among NPS (Naval Postgraduate School), NRaD (a Naval Laboratory), Purdue, and USC (University of Southern California). It builds on SmartNet, an operational scheduling framework and system for managing resources in a heterogeneous environment developed at NRaD [7]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

In summary, while the use of existing HC systems demonstrates their significant benefits, the amount of effort currently required to implement an application on an HC system can be substantial. Future research on the above open problems will improve this situation and allow HC to realize its inherent potential.

Acknowledgments - A version of some of this material also appeared in an invited keynote paper (by H. J. Siegel and M. Maheswaran) at the IX Brazilian Symposium on Computer Architectures – High Performance Computing.

References

[1] J. B. Armstrong, H. J. Siegel, W. Cohen, M. Tan, H. G. Dietz, and J. A. B. Fortes, "Dynamic task migration from SPMD to SIMD virtual machines," *1994 Int'l Conf. Parallel Processing (ICPP '94)*, Vol. II, Aug. 1994, pp. 160-169.

[2] J. B. Armstrong and H. J. Siegel, "Dynamic task migration from SIMD to SPMD virtual machines," *1st IEEE Int'l Conf. Engineering of Complex Computer Systems*, Nov. 1995, pp. 326-333.

[3] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms," *Parallel Computing*, to appear in 1998 (preliminary version in *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 96-110).

[4] M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[5] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.

[6] R. F. Freund, B. R. Carter, D. Watson, E. Keith, and F. Mirabile, "Generational scheduling for heterogeneous computing systems," *Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '96)*, Aug. 1996, pp. 769-778.

[7] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for meta-computing," *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96)*, June 1996, pp. 514-521.

[8] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[9] B. Hamidzadeh, D. J. Lilja, and Y. Atif, "Dynamic scheduling techniques for heterogeneous computing systems," *Concurrency: Practice and Experience*, Vol. 7, No. 7, Oct. 1995, pp. 633-652.

[10] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 93-100.

[11] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous computing systems," *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 135-146.

[12] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[13] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, "A case study in metacomputing: Distributed simulations of mixing in turbulent convection," *2nd Workshop on Heterogeneous Processing (WHP '93)*, Apr. 1993, pp. 101-106.

[14] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30-34.

[15] J. Rosenman and T. Cullip, "High-performance computing in radiation cancer treatment," *CRC Critical Reviews in Biomedical Engineering*, Vol. 20, 1992, pp. 391-402.

[16] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 98-117.

[17] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[18] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[19] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86-97.

[20] "Special report: Gigabit network testbeds," *IEEE Computer*, Vol. 23, No. 9, Sep. 1990, pp. 77-80.

[21] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. Parallel and Distributed Computing*, to appear in 1998 (preliminary version in *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 72-85).

[22] C. C. Weems, G. E. Weaver, and S. G. Dropsho, "Linguistic support for heterogeneous parallel processing: a survey and an approach," *3rd Heterogeneous Computing Workshop (HCW '94)*, Apr. 1994, pp. 81-88.

Heterogeneous Distributed Computing

Muthucumaru Maheswaran, Tracy D. Braun, and Howard Jay Siegel

Parallel Processing Laboratory

School of Electrical and Computer Engineering

Purdue University

West Lafayette, IN 47907-1285

USA

This is a pre-copy edited version of a chapter appearing in the *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, editor, John Wiley & Sons, New York, NY, 1999 Vol. 8, pp. 679-690.

One of the biggest challenges with high-performance computing is that as machine architectures become more advanced to obtain increased peak performance, only a small fraction of this performance is achieved on many real application sets. This is because a typical application may have various subtasks with different architectural requirements. When such an application is executed on a given machine, the machine spends most of its time executing subtasks for which they are unsuited. With the recent advances in high-speed digital communications, it has become possible to use collections of different high-performance machines in concert to solve computationally intensive application tasks. This article describes the issues involved with using such a *heterogeneous computing (HC)* suite of machines to solve application tasks.

A hypothetical example application that has various subtasks, which are best suited, for different machine architectures is shown in Figure 1 (based on (FrS93)). The example application executes for 100 time units on a baseline serial machine. The application consists of four subtasks: the first is best suited to execute on an SIMD (synchronous) parallel machine, the second is best suited for a distributed-memory MIMD (asynchronous) parallel machine, the third is best suited for a shared-memory MIMD machine, and the fourth is best suited to execute on a vector (pipelined) machine.

Executing the whole application on an SIMD machine may improve the execution time of the SIMD subtask from 25 to 0.01 time units, and the other subtasks to varying extents. The overall execution time improvement may only be about a factor of five because other subtasks may not be well suited for an SIMD machine. Using four different machines that match the computational requirements for each of the individual subtasks can result in an overall execution time that is

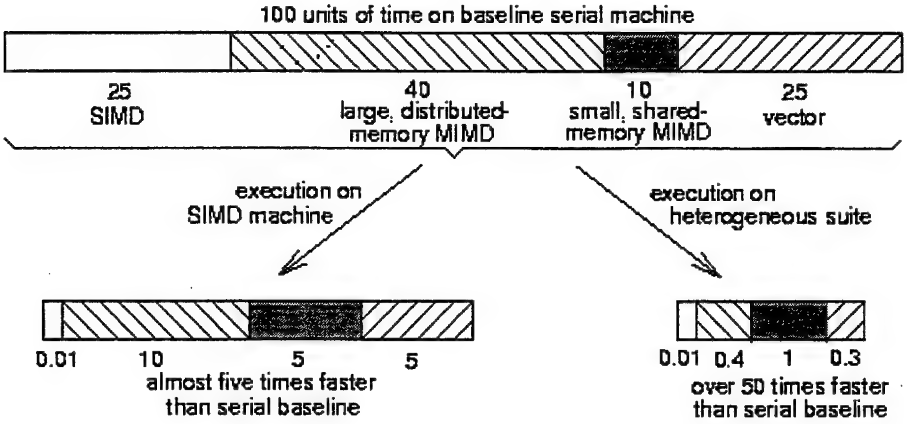

100 units of time on baseline serial machine
25
SIMD
40
large, distributed-memory MIMD
10
small, shared-memory MIMD
25
vector
execution on SIMD machine
execution on heterogeneous suite
0.01
10
5
5
almost five times faster than serial baseline
0.01
0.4
1
0.3
over 50 times faster than serial baseline

Figure 1. Hypothetical example of the advantage of using a heterogeneous suite of machines, where the heterogeneous suite time includes inter-machine communication overhead (based on (FrS93)). Not drawn to scale.

better than the baseline serial execution time by over a factor of 50. If the subtasks are dependent on any shared data, then inter-machine data transfers need to be performed when multiple machines are used. Hence, data transfer overhead has to be considered as part of the overall execution time on the HC suite. For example, in Figure 1 the time for executing on the vector machine must include any time needed to get data from the other machines.

There are many types of HC systems. This article focuses on *mixed-machine HC* systems (SiA96), where a heterogeneous suite of independent machines is interconnected by high-speed links to function as a *metacomputer* (KhP93). *Mixed-mode HC* refers to a single parallel processing system, whose processors are capable of executing in either the synchronous SIMD or asynchronous MIMD mode of parallelism, and can switch between modes at the instruction level with negligible overhead (SiM96). PASM, TRAC, OPSILA, Triton, and EXECUBE are examples of mixed-mode HC systems that have been prototyped (SiM96).

One way to exploit a mixed-machine HC environment is to decompose an application task into subtasks, where each subtask is computationally well suited to single machine architecture, but different subtasks may have different computational needs. The subtasks may have data dependencies among them. Once the subtasks are obtained, each subtask is assigned to a machine (*matching*). Then the subtasks and inter-machine data transfers are ordered (*scheduling*). It is well known that finding a matching and scheduling (*mapping*) that will minimize the overall completion time of the application is, in general, NP-complete (Fer89). Currently, programmers must manually specify the task decomposition and the assignment of subtasks to machines. One long-term pursuit in the field of heterogeneous computing is to automate this process.

In some cases, an application can be a collection of independent tasks, instead of the precedence constrained set of subtasks considered in the previous discussion. For such cases, the matching and scheduling problem considers the minimization of the completion time of the overall *meta-task* consisting of all the tasks in the application.

This article includes information that is summarized from various projects that cover different aspects of HC research. This is not an exhaustive survey of the literature. Each section of this article illustrates the concepts involved by describing a few representative techniques or systems.

In the next section, some HC application case studies are described. The section on example HC environments and tools discusses various software systems that are available to manage an HC suite of machines. Different ways of categorizing HC systems is presented in the taxonomies section. The conceptual model section provides a block diagram that illustrates what is involved in automatically mapping an application onto an HC system. Techniques for characterizing applications and representing machine performance are briefly examined in the section on task profiling and analytical benchmarking. Methods for using these characterizations in obtaining an assignment of the subtasks to machines and ordering of the subtasks assigned to each machine is explored in the section on matching and scheduling.

Example HC Application Studies

Simulation of Mixing in Turbulent Convection

An HC system at the Minnesota Supercomputer Center demonstrated the usefulness of HC through an application involving the three-dimensional simulation of mixing and turbulent convection (KIM93). The system developed for this HC application consists of a TMC SIMD CM-200 and MIMD CM-5, a vector CRAY 2, and a Silicon Graphics Inc. VGX workstation, all communicating over a high-speed *HiPPI* (*high-performance parallel interface*) network.

The necessary simulation calculations were divided into three phases: (1) calculation of velocity and temperature fields, (2) calculation of particle traces, and (3) calculation of particle distribution statistics, with refinement of the temperature field. The calculation of velocity and temperature fields associated with phase 1 is governed by two second order partial differential equations. To approximate the field components in these equations, three-dimensional cubic splines (over a grid of size 128 × 128 × 64) were used. The result was a linear system of equations representing the unknown spline coefficients. The system of equations for the spline coefficients was solved by applying a conjugate gradient method. These conjugate gradient computations were performed on the CM-5. At each time interval, the grid of 128 × 128 × 64 spline coefficients was then sent to the CRAY 2, where phase 2 was performed.

The calculation of particle traces (phase 2) involved solving a set of ordinary differential equations based on the velocity field solution from phase 1. This calculation was performed using a vectorized Lagrangian approach on the CRAY 2. Once they were computed, the coordinates of the particles and the spline coefficients of the temperature field were transferred from the CRAY 2 to the CM-200.

Phase 3 used the CM-200 to calculate statistics of the particle distribution and to assemble a three-dimensional temperature field, based on the spline coefficients received from phase 2. The 128 × 128 × 64 grid of splines was used to generate a file containing a 256 × 256 × 128 point temperature field, representing a volume of eight million voxels (a *voxel* is a three-dimensional element.) The voxels and the coordinates of the particles (one million particles were used) were then sent to the SGI VGX workstation. The SGI VGX workstation visualized the results using an interactive volume renderer. Although the simulation was a successful demonstration of the benefits of HC, the authors of (KIM93) noted that much work was still required to improve the environment for developing more HC applications.

Collision of Galaxies on the I-Way

A metacomputer consisting of a TMC MIMD CM-5, Cray MIMD T3D, IBM MIMD SP-2, and SGI Power Challenge was used to carry out a very large simulation of colliding galaxies (NoB96). The objective of this grand challenge project was to harness the power of a collection of parallel machines to address the following questions: (a) what is the origin of the large-scale structure of the universe, and (b) how do galaxies form? The simulation was performed by solving an *n*-body dynamics problem and a gas dynamics problem. The *n*-body problem was solved using the *self-consistent field (SCF)* method. The gas dynamics problem was solved by the *piecewise parabolic method (PPM)*.

The SCF code was parallelized such that if the entire calculation contains *N* particles and the computer has *P* processors, each processor evolves *N/P* particles. Each processor computes the contribution of its particles to the global gravitational field. These partial results were summed through a parallel reduction operation. After summing, the expansion coefficients were computed and broadcast to the processors. The processors then use this information to reconstruct the global gravitational field and evaluate the gravitational acceleration of the particles.

The computation for each time step in the SCF requires 36,280 FLOP/s per particle. The particles were distributed such that the computation time per time step was approximately equivalent across machines. For example, 40,960 particles per processor on the CM-5 and 57,600 particles per processor on the T3D yielded a well-balanced load. A speed of 2.5 GFLOP/s was obtained for the CM-5 and T3D suite with 6,307,840 particles, and the machines executing concurrently. The results obtained through the distributed simulation were viewed using a distributed visualization system. The SGI Power Challenge was also used for solving the *n*-body problem using the SCF code.

The PPM code was executed in parallel on an IBM SP2 machine in SPMD mode. The PPM algorithm was computationally intensive and has a high computation to communication ratio. This code obtains nearly 21.2 MFLOP/s per node on the IBM SP2.

Example HC Environments and Tools

This section overviews examples of software environments and tools that exist or are being developed for HC systems. These examples are implemented at several different levels, from the high-level management framework of SmartNet to the low-level Globus Toolkit. The functionalities described here tend to evolve and change rapidly; the descriptions here are based on the references given. Other tools include Fafner (FoF96), Legion (GrN97), Linda (CaG92), Mentat (GrW94), Ninf (SeS96), and p4 (BuL94).

SmartNet

SmartNet is a mapping framework that can be employed for managing jobs and resources in a heterogeneous computational environment (FrK96, FrG98). SmartNet enables users to execute jobs on a network of different machines as if the network was a single machine. SmartNet supports a *resource management system (RMS)* that accepts requests for mapping a job or a sequence of jobs. The jobs are assigned to the machines in the suite by the mapping algorithms built into SmartNet. Traditionally, RMSs use opportunistic load balancing schemes, where a job is assigned to the machine that becomes available first. However, SmartNet uses a multitude of more sophisticated algorithms to assign jobs to machines. SmartNet's goal is to optimize the mapping criteria in an HC environment, but these criteria are flexible, allowing SmartNet to adapt to many different situations and environments.

SmartNet exploits a variety of information resources to map and manage the applications within its heterogeneous environment. It considers (1) how well the computational capabilities of each machine match the computational needs of each application; (2) machine loading and availability; and (3) time for any needed inter-machine data transfers. SmartNet also considers the current state of other resources, such as the inter-machine communication network, before the mapping algorithms assign jobs to machines to account for the shared usage of all resources.

SmartNet can use a variety of optimization criteria to perform its mapping. Two currently implemented optimization criteria are: (1) maximizing throughput by minimizing the expected completion time of the last job, and (2) minimizing the average expected run time for each job. The mapping engine built into SmartNet uses a set of different heuristics to search the space of possible maps to find the best one, as defined by the optimization criteria. Several heuristics have been implemented. They include algorithms based on greedy strategies with varying execution time complexities, and algorithms based on evolutionary programming strategies. The mapper is modular, and is designed to implement any algorithm that satisfies relatively simple interfacing requirements. The SmartNet mapping engine considers the heterogeneity present in both the network of machines and the user tasks.

One of the advantages of SmartNet is that it does not constrain the user to a particular programming language or require special wrapper code for legacy programs. SmartNet only requires the user to provide a description of the time complexity of each program. SmartNet demonstrates that the performance of a metacomputer can be enhanced by considering both the machine loading and heterogeneity in coordinating the execution of user programs. Thus, SmartNet provides a global, general-purpose, scalable, and tunable resource management framework for HC systems. SmartNet was designed and developed at NRaD (a Naval laboratory), and is operational at several research laboratories.

Ideas and lessons learned from SmartNet are used in designing and implementing the DARPA/ITO Quorum Program project called *MSHN* (*Management System for Heterogeneous Networks*). MSHN is a collaborative research effort among NPS (Naval Postgraduate School), NRaD (a Naval Laboratory), Purdue University, and USC (University of Southern California). The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

NetSolve

NetSolve is a client-server-based application designed to provide network access to remote computational resources for solving computationally intense scientific problems (CaD97). The machines participating in a NetSolve system can be on a local or geographically distributed HC network.

For a given problem, a NetSolve client (i.e., and application task) sends a request to a NetSolve agent (residing in the same or different machine). The NetSolve agent then selects a resource for the problem based on the size and nature of the problem. There can be several instantiations of NetSolve agents and clients. Every machine in a NetSolve system runs a NetSolve computational server for access to the machine's scientific packages. The NetSolve system can be accessed from a variety of interfaces, including MATLAB, shell scripts, C, and FORTRAN. NetSolve can also be called in a blocking or nonblocking fashion, so that computations can be performed concurrently on the client system, thus improving performance.

NetSolve uses load balancing to improve system performance. For every machine in the NetSolve system, the execution time for a given problem is estimated. This estimate is used to determine the hypothetical best machine on which to execute the problem. This execution time estimate is based on several factors, including size of the data, size of the problem, complexity of the algorithm, network parameters, and machine characteristics.

To maintain accurate system performance information, each instance of an agent maintains a value of the workload from every other server. A new workload value is conditionally broadcast at regular intervals, i.e., if the value is outside a defined range, then the server broadcasts the value. This allows accurate system information to be maintained, without needlessly burdening the network with the same workload value.

NetSolve has capabilities for handling fault tolerance at several different levels. Servers generally handle failure detection. Clients minimize side effects from service failures by maintaining lists of computational servers. Future work includes increasing the number of interfaces, improved load balancing, and allowing user-defined functions.

PVM and HeNCE

Parallel Virtual Machine (*PVM*) is a software environment that enables an HC system to be utilized as a single, connected, flexible, and concurrent computational resource (BeD93, Sun90). The PVM software package consists of system-level daemons, called *pvmds*, which reside on each machine in the HC system, and a library of PVM interface routines.

The pvmds are responsible for providing services to both local processes and remote processes executing on other machines in the HC system. By considering the entire set of pvmds collectively, a virtual machine is formed. This virtual machine allows the HC system to be viewed as a single metacomputer. The pvmds provide three major services: process and virtual machine management, communication, and synchronization. Process and virtual machine management issues include: computational unit scheduling and placement, configuration and inclusion of remote computers into the virtual machine, and naming and addressing of resources. Communication is performed with asynchronous message passing, allowing a sending process to continue execution without having to wait for a receive acknowledgment. The synchronization among processes provided by the pvmds can be accomplished with barriers or other techniques.

Multiple processes can be synchronized, including synchronization of processes that are executing on a local machine and processes that are executing remotely.

The PVM system also provides a library of interface routines. Applications access platforms in the HC system via library calls embedded within imperative procedural languages such as C or FORTRAN. The library routines and the pvmds (resident on each machine) interact to provide communication, synchronization, and process management services. A single pvmd may provide the requested service, or the service can be provided by a group of pvmds in the HC system working in concert.

The *heterogeneous network computing environment* (*HeNCE*) is a tool that aids users of PVM in decomposing their application into subtasks and deciding how to distribute these subtasks to the machines currently available in the HC system (BeD93). HeNCE allows the programmer to explicitly specify the parallelism for an application by creating a directed graph, where nodes represent subtasks (written in either FORTRAN or C) and arcs represent precedence constraints and flow dependencies. HeNCE also has four types of control constructs: conditional, looping, fan out, and pipelining.

The cost of executing each subtask on each machine in the HC system is represented by a user specified cost matrix. The meaning of the parameters within the cost matrix is defined by the user (e.g., estimated execution times or utilization costs in terms of dollars). At execution time, HeNCE uses the cost matrix to estimate the most cost effective machine on which to execute each subtask.

Once the directed graph and cost matrix are specified, HeNCE uses PVM constructs to configure a subset of the machines defined in the cost matrix as a virtual machine. Then HeNCE initiates execution of the program. Each subtask in the graph is realized by a distinct process on some machine in the HC system. The subtasks communicate by sending parameter values necessary for execution of a given subtask. These parameter values are specified by the user for each subtask. Parameter values needed to begin execution of a subtask are obtained from predecessor subtasks. If the set of immediate predecessor subtasks does not have all of the required parameters for a subtask to begin execution, earlier predecessor subtasks are checked until all of the required parameters are located. Once all of the parameters are found, the subtask is executed, and the appropriate parameters are passed onto descendant subtasks. HeNCE can trace the execution of the application for the display in real time or replay later.

Globus Metacomputing Infrastructure Toolkit

The Globus project (FoK97, FoK98) defines a set of low-level mechanisms that provide basic HC infrastructure requirements, such as communication, resource allocation, and data access. These low-level mechanisms are part of the Globus metacomputing infrastructure toolkit, and can be used to implement higher level HC services (e.g., mappers and parallel programming tools).

Each component in the toolkit defines an interface and an implementation for any HC environment. The interfaces allow higher level services to invoke that component's mechanisms.

The implementation uses low-level instructions to realize these mechanisms on the different systems occurring within HC environments. Presently, the Globus toolkit consists of six components. (1) The *communication* component provides a wide range of communication methods, including message passing, remote procedure call, distributed shared memory, and multicast. (2) The *resource location and allocation* module provides mechanisms for expressing application resource requirements, identifying resources suitable for these requirements, and scheduling these resources after they have been located. (3) In the *unified resource information service* component, a mechanism is provided for posting and receiving real-time information about the HC environment. (4) The *data access* module is responsible for providing high-speed access to remote data and files. (5) Once a resource has been allocated, the *process creation* component is used to initiate computation. This includes initialization of executables, starting an executable, passing arguments, integrating the new process into the rest of the computation, and process termination. Finally, (6) The *authentication interface* module provides basic authentication mechanisms for validating the identity of both users and resources.

The modules of the Globus toolkit can be considered to define an abstract HC system. The definition of this HC system simplifies development of higher level applications by allowing HC programmers to think of geographically distributed, heterogeneous collections of resources as unified entities. It also allows for a range of alternative infrastructures, services, and applications to be developed. The stated long-term goal of the Globus project is to address the problems of configuration and performance optimization in HC environments. To accomplish this goal, the Globus project is designing and constructing a set of higher level services layered on the Globus toolkit. These higher level services would form an adaptive wide area resource environment (AWARE).

Taxonomies of Heterogeneous Computing

One of the first classifications of HC systems, provided in (WaS93), divides systems into either *mixed-machine HC* systems or *mixed-mode HC* systems. These two categories were defined earlier in this article. Mixed-machine HC systems denote *spatial* heterogeneity, whereas mixed-mode HC systems denote *temporal* heterogeneity. Recently, researchers have further refined this classification to obtain different schemes.

In (EkT96), a taxonomy called the EM^3 (EMMM = *e*xecution *m*ode, *m*achine *m*odel) is presented for HC systems. In this scheme, HC systems are categorized in two orthogonal directions. One direction is the *execution mode* of the machine, which is defined by the type of parallelism supported by the machine. For example, high performance computing architectures are often specialized to support either MIMD, SIMD, or vector execution modes. The heterogeneity based on this criterion can be temporal or spatial. The second categorization is the *machine model,* which is defined as the machine architecture and machine performance. For example, Sun Sparc CY7C601 and Intel i860 are considered different architectures. In addition, two CPUs of the same type but driven by different speed clocks provide different machine performance and hence are considered different machine models. The heterogeneity based on this criterion is always spatial in nature.

HC systems are classified by counting the number of *execution modes (EM)* and the number of *machine models (MM)*. The four categories proposed in (EkT96) are (a) *SESM* (single execution mode, single machine model), (b) *SEMM* (single execution mode, multiple machine model), (c) *MESM* (multiple execution mode, single machine model), and (d) *MEMM* (multiple execution mode, multiple machine model). Fully homogeneous systems make up the SESM class. HC systems composed of different architectures (or clock speeds) with the same execution mode are in the SEMM class. Both the SEMM and MEMM classes are mixed-machine systems, but only the MEMM class can include different execution models and mixed-mode machines. The MESM corresponds to mixed-mode systems, i.e., temporal heterogeneity. HC systems composed of different architectures, where some of the machines use different execution models fall into the MEMM class.

In the classification provided in (Esh96), HC systems are grouped into: (a) system heterogeneous computing (*SHC*) and (b) network heterogeneous computing (*NHC*). SHC is further divided into multimode SHC and mixed-mode SHC. *Multimode SHC* systems can perform computations in both SIMD and MIMD modes simultaneously, and exhibit spatial heterogeneity in a single machine. *Mixed-mode SHC* systems switch execution between the SIMD and MIMD modes of parallelism, exhibit temporal in a single machine. The NHC systems are divided into multimachine NHC and mixed-machine NHC. *Multimachine NHC* denotes homogeneous distributed computing systems and *mixed-machine NHC* indicates heterogeneous distributed computing systems.

A Conceptual Model of Heterogeneous Computing

In the examples featured in the application studies section, the programmer specified the machine assignment for each program segment and initial data item. One of the long-term goals of HC research is to develop software environments that will automatically find a near-optimal mapping for an HC program expressed in a machine-independent high-level language. Performing the mapping automatically has the following benefits: (1) an increase in portability because the programmer need not be concerned with the composition of the HC suite, (2) easier use of the HC system, and (3) the possibility of deriving better mappings than the user can with ad hoc methods. While no such environment exists today, many researchers are working towards developing an environment to automatically and efficiently perform the mapping of subtasks to machines in an HC suite. A conceptual model for such an environment using a dedicated HC suite of machines is described in Figure 2 (based on (SiD97) and (MaB98)).

For stage 1, information about the type of each application task and each machine in the HC suite is used to generate a set of parameters relevant to both the computational characteristics of the applications and the machine architecture features of the HC system. From this set of parameters, categories for computational requirements and categories for machine capabilities are derived.

Stage 2 consists of two components, task profiling and analytical benchmarking. *Task profiling* decomposes the application task into subtasks, where each subtask is computationally homogeneous. Usually, different subtasks will have different computational needs. The computational requirements of each subtask are quantified by profiling the code and data. *Analytical benchmarking* quantifies how effectively each of the machines available in the suite

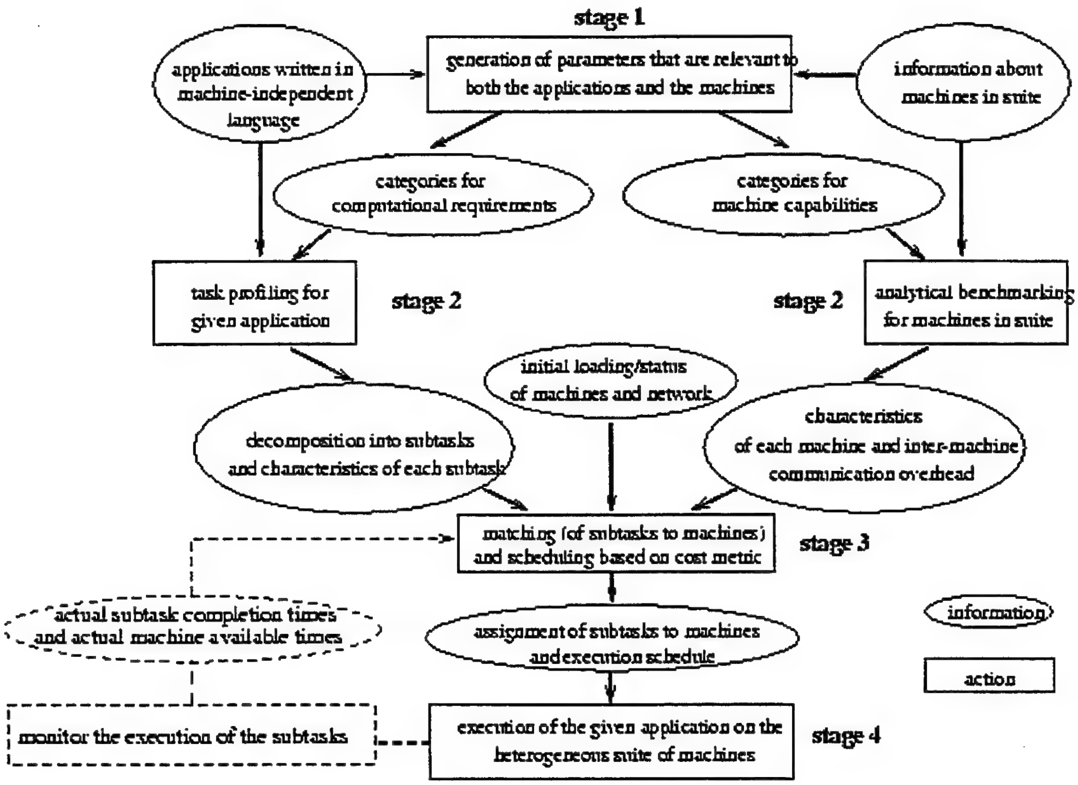

stage 1
applications written in machine-independent language
generation of parameters that are relevant to both the applications and the machines
information about machines in suite
categories for computational requirements
categories for machine capabilities
task profiling for given application
stage 2
stage 2
analytical benchmarking for machines in suite
initial loading/status of machines and network
decomposition into subtasks and characteristics of each subtask
characteristics of each machine and inter-machine communication overhead
matching (of subtasks to machines) and scheduling based on cost metric
stage 3
actual subtask completion times and actual machine available times
assignment of subtasks to machines and execution schedule
information
action
monitor the execution of the subtasks
execution of the given application on the heterogeneous suite of machines
stage 4

Figure 2. Model for integrating the software support needed for automating the use of heterogeneous computing systems (based on (SiD97) and (MaB98)).

performs on each of the types of computations required. The components of stage 2 are discussed further in the next section.

Stage 3 requires the information available from stage 2 to derive the estimated execution time of each subtask on each machine in the HC suite, along with the associated inter-machine communication overheads. These statically derived results are then incorporated with initial values for machine loading, inter-machine network loading, and status parameters (e.g., machine/network faults) to perform the matching and scheduling of subtasks to machines. The result is an assignment of subtasks to machines and an execution an execution schedule based on certain cost metrics (e.g., minimizing the overall execution time for all tasks.) Matching and scheduling in HC systems is examined in more detail later in this article.

Stage 4 is the execution of the given application. If a dynamic matching and scheduling system is employed, the subtask completion times and loading/status of the machines/network are monitored. The monitoring process is necessary because the actual computation times and data transfer times may be input-data dependent and deviate from the static estimates. This information may be used to re-invoke the matching and scheduling of stage 3 to improve the machine assignment and execution schedule. Automatic HC is a relatively new field. Preliminary frameworks for task profiling, analytical benchmarking, and mapping have been proposed, however, further research is needed to make this conceptual model a reality (SiA96, SiD97).

Task Profiling and Analytical Benchmarking

Task profiling specifies the types of computations that are present in the application program by decomposing the source program into homogeneous *code blocks* based on computational requirements (Fre89). The set of code types defined is based on the features of the machine architectures available and the processing requirements of the applications being considered for execution on the HC system (phase 1 of the conceptual mode described in the previous section). This set of code types will be a function of both the application task code and the types and sizes of data sets it is to process. Task profiling is performed in stage 2 of the conceptual model presented in the previous section.

Analytical benchmarking provides a measure of how well each of the available machines in the heterogeneous suite performs on each of the given code types (Fre89). In combination, task profiling and analytical benchmarking provide the necessary information for the matching and scheduling step (discussed in the next section). The performance of a particular code type on a specific kind of machine is a multi-variable function. The variables within this performance function can include the following: the requirements of the application (e.g., data precision), the size of the data set to be processed, the algorithm to be applied, programmer and compiler efforts to optimize the program, and the operating system and architecture of the machine that will execute the specific code type (GhY93).

Selection theory is a collection of mathematical formulations that have been proposed for selecting the most appropriate machine for each code block. Many formulations (e.g., (ChE93, KhP93, WaK92)) define analytical benchmarking as a method of measuring the optimal speedup of a particular machine type executing the best-matched code type to a baseline system. The ratio between the actual speedup and the optimal speedup defines how well a code block is matched with each machine type. Generally, the actual speedup is less than the optimal speedup.

The *parallel assessment window system* (*PAWS*) and the *distributed heterogeneous supercomputing management system* (*DHSMS*) are briefly examined here. They represent example preliminary frameworks for implementing task profiling and analytical benchmarking.

The PAWS prototype consists of four tools: the application characterization tool, the architecture characterization tool, the performance assessment tool, and the interactive graphical display tool (PeG91). First, the *application characterization tool* transforms a given program written in a specific subset of Ada into an acyclic graphical language that illustrates the program's data dependencies. The tool groups sets of nodes and edges into functions and procedures; allowing the execution behavior of a given program to be described at various levels. However, this tool does not perform task decomposition based on computational requirements and machine capabilities.

To benchmark machines, the *architecture characterization tool* divides the architecture of a specific type of machine into four categories: computation, data movement and communication, I/O, and control. Each category can be repeatedly partitioned into subsystems, until the lowest level subsystems can be described by raw timing information. The *performance assessment tool* uses the information from the architecture characterization tool to generate timing information for operations on a given machine. Two sets of performance parameters for an application, parallelism profiles and execution profiles, are generated by the performance assessment tool. *Parallelism profiles* describe the applications' theoretical upper bounds of performance (e.g., the maximal number of operations that can be parallelized). *Execution profiles* represent the estimated performance of the applications after they have been partitioned and mapped onto one particular machine. Both parallelism and execution profiles are produced by traversing the applications' task-flow graph and then computing and recording each node's performance and statistically based execution time estimates. The *interactive graphical display tool* is the user interface for accessing all the other tools in PAWS.

The DHSMS classifies task profiling and analytical benchmarking results within a systematic framework (GhY93). First, DHSMS generates a *universal set of codes* (*USC*) for task profiling. The USC can be considered as a standardized set of benchmarking programs used in analytical benchmarking. Similar to the hardware organizational information maintained by the architectural characterization tool in PAWS, a USC is constructed using a hierarchical structure based on the machines in the HC suite. At the highest level of this hierarchical structure, modes of parallelism are selected to specify the machine architectures. At the second level, finer architectural characteristics, such as the organization of the memory system, can be chosen. This hierarchical structure is organized so that the architectural characteristics at any level are choices for a given category (e.g., type of interconnection network used). DHSMS assigns a *code type* (i.e., computational characteristic) to each path from the root of the hierarchical structure to a leaf node. Every such path represents a specific set of architectural features, defined by the nodes within the path.

The DHSMS approach is extended in (YaA93) to include the generation of a *representative set of templates* (*RST*) that can characterize the execution behavior of the programs at various levels of detail. Many HC methodologies include mathematical formulation for task profiling and analytical benchmarking that is similar in concept to that used in DHSMS (e.g., (ChE93, Fre89, NaY94, WaK92)).

Matching and Scheduling

Overview

Matching and scheduling is an important component of the conceptual model of the automatic HC presented earlier. Finding an optimal solution for the matching and scheduling problem is NP-complete (Fer89). For example, consider matching and scheduling 30 subtasks onto five machines. This means that there are 5^{30} possible mappings. Assuming it takes only one nanosecond to evaluate the quality of one mapping, an exhaustive comparison of all possible mappings would require 5^{30} nanoseconds $> 4 \times 10^{10}$ seconds > 1000 years! Therefore, it is necessary to have heuristics to find the best mappings rather than evaluate all possible mapping combinations. Mapping schemes can be either *static*, where the mapping decisions are made off-line before the execution of the subtask (e.g., (EsW96, KaA97, ShW96, SiL93, SiY96, WaS97)) or *dynamic*, where the mapping decisions are made on-line during the execution of the subtasks (e.g., (FrC96, HaL95, LeP95, MaS98)).

A Mathematical Formulation of Matching and Scheduling in HC

The *optimal selection theory* (*OST*) (Fre89) provides the first known mathematical formulation for selecting an optimal heterogeneous configuration of machines for a given set of problems under a fixed cost constraint in HC systems. In the OST, it is assumed that the application consists of non-overlapping *code segments* that are totally ordered in time. The overall execution time of the application equals the sum of the execution times of its code segments.

A code segment is defined to be *decomposable* if it can be partitioned further into *code blocks* that can be executed in multiple copies of the best-matched machine type. A sufficient number of machines of the best-matched machine type are assumed to be available. For simplicity, linear speedup is assumed for a decomposable code segment. Let the application have $S \geq 1$ code segments and $M \geq 1$ different types of machines to execute the code segments. Let v_j be the number of machines of type j and the cost of using a machine of type j is c_j. The estimated execution time of code segment i on machine type j is given by $t_{i,j}$, for all $1 \leq i \leq S$, $1 \leq j \leq M$. The optimization problem involves minimizing the total execution time of the application, T, defined below, subject to a given constraint on the total cost of the machines used, C. The cost incurred by using type j machines is given by v_jc_j. Assume that code segment i is best suited on machine type j. Because there are v_j number of type j machines the execution time of code segment i on this type of machine is $t_{i,j}/v_j$. Thus, the goal is to minimize the total execution time of the application:

$$\text{minimize} \qquad T = \sum_{i=1}^{S} \left\{ \frac{t_{i,j}}{v_j} \right\}$$

given the total cost constraint:

$$\sum_{j=1}^{M} v_j c_j \leq C$$

The *augmented optimal selection theory* (*AOST*) (WaK92) is an extension of the OST. The AOST considers the performance of the code segments for all available machine type choices (not just the best-matched machine type) and a fixed number of machines of each type. In practice, this extension is useful because the best-matched machine may not be available, and only a limited number of machines of each type may be available. Another extension of the OST is provided by the *heterogeneous optimal selection theory* (*HOST*) (ChE93). The HOST extends AOST by allowing concurrent execution of mutually independent code segments on different types of machine and incorporating the effects of different possible local mappings. Consider an example code block for the multiplication of two matrices onto a distributed memory parallel machine. Many implementations with varying execution characteristics can be derived for this code block. The HOST assumes that the best mapping choice (minimum execution time) is known for each code block.

The *generalized optimal selection theory* (*GOST*) further refines the OST to handle communication delays (NaY94). In the GOST, the basic code element is called a *process*, which is *nondecomposable*. The application is represented by a directed acyclic graph (DAG), where a node denotes a process and an arc denotes a dependency between two processes. A node has a number of weights attached to it, corresponding to the execution times of the process on each machine type for each known mapping onto that machine. An edge has a number of weights, one for each communication path between each possible pair of host machines. In (NaY94), a matching and scheduling problem is formulated with the objective of assigning each node to a machine type and finding a start time for each node so that the overall completion time of the application can be minimized. Polynomial-time algorithms are provided in (NaY94) for certain types of DAGs.

Static Matching and Scheduling Heuristics

The heuristics summarized below are based on the following assumptions, unless noted otherwise. Each application task is represented by a DAG, whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies between subtasks. Each edge is labeled by the global data item that is transferred between the subtasks connected by the edge. The matching and scheduling algorithm controls the HC machine suite (hardware platform). Subtask execution is non-preemptive. The estimated expected execution time of each subtask on each machine is known. For each pair of machines in the HC suite, an equation for estimating the time to send data between those machines as a function of data set size is known.

Cluster-M Mapping Heuristic

The HC matching and scheduling process can be thought of as a mapping of a graph that represents a set of subtasks (*task graph*) onto a graph that represents the set of machines in the HC suite (*system graph*) (Esh96). In Cluster-M, the mapping is performed in two stages. In the first stage, the task graph and system graph are clustered. The task graph clustering combines the communication intensive subtasks into the same cluster. Similarly, the system graph clustering combines the machines that are tightly coupled (i.e., small inter-machine communication times) into the same cluster. The clustering of the task graph does not depend on the clustering of the system graph and vice-versa. Therefore, a task or system graph needs to be clustered only once. In the second phase, the clustered task graph is mapped onto a clustered system graph. The clustering reduces the complexity of the mapping problem and improves the quality of the resulting mapping.

The Levelized Min-Time Heuristic

The *levelized min time (LMT)* heuristic is a static matching and scheduling algorithm for subtasks in an HC system (IvO95). It is based on list-scheduling class of algorithms. The LMT algorithm uses a two-phase approach. The first phase uses a technique called level sorting to order the subtasks based on the precedence constraints. The level sorting can be defined as follows. The level 0 contains subtasks with no incident arcs. All predecessors with arcs to a level *k* subtask are in levels *k*–1 to 0. For each subtask in level *k* there exists at least one incident arc (data dependency) such that the source subtask is in level *k*–1. The level sorting technique clusters subtasks that are able to execute in parallel.

The second phase of the LMT algorithm uses a min time algorithm to assign the subtasks level by level. The min time algorithm is a greedy method that attempts to assign each subtask to the best machine. If the number of subtasks is more than the number of machines, then the smallest subtasks are merged until the number subtasks is equal to the number of machines. Then the subtasks are ordered in descending order by their average computation time. Each subtask is assigned to the machine with the minimum completion time. Sorting the subtasks by the average computation time increases the likelihood of larger subtasks getting faster machines.

One optimization to the LMT algorithm discussed in (IvO95) involves the use of information on the amount of communication between subtasks in different levels. This enables the scheduler to map subtasks that share large amounts of data to the same machine.

Genetic Matching and Scheduling Heuristic

In *genetic algorithms (GAs)*, some of the possible solutions are encoded as *chromosomes*, the set of which is called as a *population*. This population is iteratively operated on by the following steps until a stopping criterion is met. The first step is the selection step, where some chromosomes are removed and others duplicated based on their *fitness value* (a measure of the quality of the solution represented by a chromosome). This is followed by the crossover step, where some chromosomes are paired and the corresponding components of the paired chromosomes are exchanged. Then, the chromosomes are randomly mutated, with the constraint that the resulting chromosomes still represent valid solutions for the physical problem.

To apply GAs to the subtask matching and scheduling problem in HC systems using the approach presented in (WaS97), the chromosomes are encoded with two parts: the matching string (*mat*) and the scheduling string (*ss*). If mat(i) = j, then subtask s_i is assigned to machine m_j. The scheduling string is a topological sort of the DAG representing the task (i.e., a valid total ordering of the partially ordered DAG). If ss(k) = i, then subtask s_i is the k-th subtask in the total ordering. Each chromosome is associated with a fitness value, which is the completion time of the solution represented by this chromosome (i.e., the expected execution time of the application task if the mapping specified by this chromosome were used).

On small-scale tests with up to ten subtasks, three machines, and population size of 50, the GA approach found a solution (mapping) that had the same expected completion time as the optimal solution found by exhaustive search. On large-scale tests with up to 100 subtasks, 20 machines, and a population size of 200, the GA approach produced solutions (mappings) that were on the average 150% to almost 300% better than those produced by the (faster) non-evolutionary basic levelized min-time (*LMT*) heuristic proposed in (IvO95).

Dynamic Matching and Scheduling Heuristics

The static mapping heuristics assume that accurate estimates are available for parameters such as subtask completion times and data transfer times. However, in general, such estimates have a degree of uncertainty in them because subtask computation times and data transfer times may be dependent on input data. Therefore, dynamic mapping heuristics that can handle the uncertainty may be needed. Researchers have proposed different dynamic heuristics for varying HC models (e.g., (BuR98, FrC96, HaL95, LeP95)). Furthermore, in dynamic mapping heuristics machines can come on-line and go off-line at run time.

Hybrid Remapper

The hybrid remapper heuristic described here is a dynamic algorithm for matching and scheduling subtask DAGs onto HC systems (MaS98). An initial, statically obtained matching and scheduling is provided as input to the hybrid remapper. The hybrid remapper executes in two phases. In the first phase of the hybrid remapper, performed prior to application execution, the subtasks are partitioned into L levels as in the LMT heuristic. Each subtask is assigned a rank by examining the subtasks from level $L-1$ to level 0. The *rank* of each subtask in the $(L-1)$-th level is set to its expected computation time on the machine to which it was assigned by the initial matching. The rank of a subtask s_i in level k is determined by computing the length of the critical path from s_i to the subtask where the execution terminates.

The second phase of the hybrid remapper occurs during the application execution. The hybrid remapper changes the matching and scheduling of the subtasks in level k while the subtasks in level $(k-1)$ or before are running. The subtasks in level k are examined in descending order of the static rank and each subtask is assigned to a machine with the earliest completion time for that particular subtask. The hybrid remapper starts scheduling the level k subtasks when the first level $(k-1)$ subtask begins its execution, and must finish the level k remapping before any level k subtask has the input data and machine available it needs to execute. When level k is being scheduled, it is highly likely that actual execution time information can be used for many subtasks from levels 0 to $(k-2)$. There may be some subtasks from levels 0 to $(k-2)$ that are still running or waiting execution when subtasks from level k are being considered for remapping. For such subtasks, expected execution times are used.

Simulation results indicate that the hybrid remapper can improve the performance of a statically obtained initial matching and scheduling by as much as 15% for some cases. Initial mappings for the simulation were generated using the baseline heuristic (WaS98). The timings also indicate that the remapping time needed per level of subtasks is on the order of hundreds of milliseconds for up to 50 machines and 500 subtasks. In the worst case situation, to obtain complete overlap between the execution of the subtasks and the operation of the hybrid remapper, the computation time for the shortest running subtask must be greater than the per level scheduling time. Ongoing research will examine ways to increase the performance gain obtained from the use of the hybrid remapper.

Generational Scheduling

The *generational scheduling* (*GS*) heuristic is a dynamic mapping heuristic for subtasks in HC systems (FrC96). It is a cyclic heuristic with four stages. First, the GS forms a partial scheduling problem by pruning all the subtasks with unsatisfied precedence constraints from the initial DAG that represents the application. That is, the initial partial scheduling problem consists of subtasks that are either independent or have no incident edges in the DAG. The subtasks in the initial partial scheduling problem are then mapped onto the machine using an auxiliary scheduler. The auxiliary scheduler considers the subtasks for assignment in a first come first serve order. A subtask is assigned to a machine that minimizes the completion time of that particular subtask.

When a subtask from the initial partial scheduling problem completes its execution, the GS heuristic performs a remapping. During the remapping, the GS revises the partial scheduling problem by adding and removing subtasks from it. The completion of the subtask that triggered the remapping event may have satisfied the precedence constraints of some subtasks. These subtasks are added to the initial partial scheduling problem. The subtasks that have already started execution are removed from the initial partial scheduling problem. Once the revised partial scheduling problem is obtained, the subtasks in it are mapped on to the HC machine suite using the auxiliary scheduler. This procedure is cyclically performed until the completion of all subtasks.

Self-Adjusting Scheduling for Heterogeneous Systems

The *self-adjusting scheduling for heterogeneous systems* (*SASH*) heuristic is a dynamic scheduling algorithm for mapping a set of independent tasks (meta-task) onto an HC suite of machines (HaL95). One processor is dedicated to compute the schedule, and this scheduling is overlapped with the execution of the tasks. At the end of each scheduling phase, the scheduling processor loads the tasks in that phase onto the working processors' local queues. The dedicated processor then schedules the next subset of tasks while the previously scheduled tasks are being executed by the working processors.

The duration of the scheduling phase is determined by a lower-bound estimate of the load on the working processors. The first working processor to complete executing all of the tasks in its local queue signals the scheduling processor, and the scheduling processor then assigns more tasks to all processors based on the partial schedule just computed. The SASH heuristic computes the schedules using a variation of the branch-and-bound algorithm. In this variation, a tree is used to represent the space of possible schedules. A node in the tree represents a partial schedule consisting of a set of tasks assigned to a corresponding set of processors. An edge from a node represents an augmentation of the schedule by one more task-to-processor assignment.

A scheduling phase consists of one or more SASH iterations. In an iteration, the node with the lowest cost is expanded by augmenting the partial schedule with another task-to-processor assignment. The node expansions terminate when all the tasks are scheduled or the time for scheduling phase *i* expires.

Matching and Scheduling Meta-Tasks

As defined earlier in this article, a meta-task is a collection of independent tasks that need to mapped onto an HC suite. Some tasks may have subtasks with data dependencies among them. Most of the heuristics and environments considered in the previous sections of this article are suitable for mapping tasks that can be decomposed into subtasks with data dependencies. Exceptions include the environments SmartNet and NetSolve (which can manage meta-tasks and decomposed tasks), and the mapping heuristic SASH (which was for meta-tasks).

Typically, when independent tasks are involved, the tasks arrive at the HC suite in a random fashion for service. Also, some machines in the suite may go off-line or new machines may come on-line. Therefore, dynamic mapping heuristics are usually employed to assign the tasks to machines. Furthermore, the tasks can have deadlines and priorities associated with them. Two types of dynamic approaches are on-line and interval. On-line approach assigns each task to a machine when it is submitted. Interval approach waits for a set of new tasks to arrive and then map those tasks and remap any earlier tasks that have not yet started execution. Developing heuristics for matching and scheduling meta-tasks in HC systems is an active research area.

Summary and Future Directions

This article illustrates the concepts involved in heterogeneous distributed computing by sampling various research and development activities in this area. It is by no means an exhaustive survey of the HC literature. The practical importance of HC is revealed by the application studies summarized in this article. The conceptual model provided in Figure 2 envisions an automatic HC programming environment. Most components of the model require further research for devising practical and theoretically sound methodologies (KhP93, SiA96, SiD97). A flavor of the work performed in matching and scheduling is also provided in this article.

An important question that is particularly relevant to stages 1 and 2 of the conceptual model is: "What information can be obtained automatically and what information should be provided by the programmer?" Following areas should be further researched to realize the automatic HC environment envisioned in Figure 2: (1) developing machine-independent programming languages, (2) designing high-speed networking systems, (3) studying communication protocols for reliable, low overhead data transmission with given quality of service requirements, (4) devising debugging tools, (5) formulating algorithms for task migration, fault tolerance, and load balancing, (6) designing user interfaces and user friendly programming environments, and (7) developing algorithms for applications with heterogeneous computing requirements. Most of these issues pertain to meta-tasks as well as an application that is decomposed into subtasks.

Machine-independent programming languages (WeW94) that allow the user to augment the code with compiler directives are necessary to program the HC system. Following aspects should be considered in designing the language and directives: (a) the compilation of the program into efficient code for the machines in the suite, (b) the decomposition of tasks into subtasks, (c) the determination of computational requirements of each subtask, and (d) the use of machine dependent subroutine libraries.

There is a need for debugging and performance tuning tools that can be used across an HC suite of machines. This involves research in the areas of distributed programming environments and visualization techniques.

Another area of research is dynamic task migration between different parallel machines at execution time. Current research in this area involves determining how to move an executing task between different machines (ArS94, ArS95) and how to use dynamic task migration for load rebalancing or fault tolerance.

Ideally, information about the current loading and status of the machines in the HC suite and the network should be incorporated into the mapping decisions. Methods must be developed for measuring the current loading, determining the status (e.g., faulty or not faulty), and estimating the subtask completion times. Also, the uncertainty present in the estimated parameter values such as subtask completion times should be taken into consideration in determining the machine assignment and execution schedule.

In summary, while the use of currently available HC systems demonstrate their significant benefits, most of them require the programmer to have an intimate knowledge of what is involved in mapping the application task(s) onto the suite of machines. Hence, the widespread use of HC system is hindered. Further research on the areas briefly explained in this article should improve this situation and allow HC to realize its full potential.

Acknowledgment: The development of this article was supported in part by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-97-M-0900 and N62271-98-M-0217. The authors thank Noah Beck, Ricky Kwok, and Mitch Theys for their comments.

Bibliography

[ArS94] J. B. Armstrong, H. J. Siegel, W. Cohen, M. Tan, H. G. Dietz, and J. A. B. Fortes, Dynamic task migration from SPMD to SIMD virtual machines. *1994 Int. Conf. Parallel Processing (ICPP '94)*, Vol II, Aug. 1994, pp. 160-169.

[ArS95] J. B. Armstrong and H. J. Siegel, Dynamic task migration from SIMD to SPMD virtual machines. *1st IEEE Int. Conf. Engineering of Complex Computer Systems*, Nov. 1995, pp. 326-333.

[BeD94] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, Visualization and debugging in a heterogeneous environment. *Computer*, **26** (6): 88-95, 1993.

[BuR97] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms. *The Supercomputing Journal*, to appear in 1998 (preliminary version in *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 96-110).

[BuL94] R. M. Butler and E. L. Lusk, Monitors, messages, and clusters: The p4 parallel programming system. *Parallel Computing*, **20**: 547-564, 1994.

[CaG92] N. Carriero, C. Gelernter, and T. G. Mattson, Linda in heterogeneous computing environments. *1st Workshop on Heterogeneous Processing (WHP '92)*, Mar. 1992, pp. 43-46.

[CaD97] H. Casanova and J. Dongarra, NetSolve: A network-enabled server for solving computational science problems. *The Int. J. Supercomputer Applications and High Performance Computing*, **11** (3): 212-223, 1997.

[ChE93] S. Chen, M. M. Eshaghian, S. Khokhar, and M. E. Shaaban, A selection theory and methodology for heterogeneous supercomputing. *2nd Workshop on Heterogeneous Processing (WHP '93)*, Apr. 1993, pp. 15-22.

[EkT96] I. Ekemecic, I. Tartalja, and V. Milutinovic, A survey of heterogeneous computing: Concepts and systems. *Proc. IEEE*, **84**: 1127-1144, 1996.

[Esh96] M. M. Eshaghian (ed.), *Heterogeneous Computing*. Norwood, MA: Artech House, 1996.

[EsW96] M. M. Eshaghian and Y.-C. Wu, A portable programming model for network heterogeneous computing. In M. M. Eshaghian (ed.), *Heterogeneous Computing*, Norwood, MA: Artech House, pp. 155-195, 1996.

[Fer89] D. Fernandez-Baca, Allocating modules to processors in a distributed system. *IEEE Trans. Softw. Eng*, **SE-15**: 1427-1436, 1989.

[FoK97] I. Foster and C. Kesselman, Globus: A metacomputing infrastructure toolkit. *The Int. J. Supercomputer Applications and High Performance Computing*, **11** (2): 115-128, 1997.

[FoK98] I. Foster and C. Kesselman, The Globus project: A status report. *7th Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 4-18.

[FoF96] G. Fox and W. Furmanski, Web based high performance computing and communications. *5th IEEE Symp. High Performance Distributed Computing*, Aug. 1996.

[Fre89] R. F. Freund, Optimal selection theory for superconcurrency. *Supercomputing '89*, Nov. 1989, pp. 699-703.

[FrC96] R. F. Freund, B. R. Carter, D. Watson, E. Keith, and F. Mirabile, Generational scheduling for heterogeneous computing systems. *Int. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '96)*, Aug. 1996, pp. 769-778.

[FrG98] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet. *7th Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184-199.

[FrK96] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, SmartNet: A scheduling framework for meta-computing. *2nd Int. Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96)*, June 1996, pp. 514-521.

[FrS93] R. F. Freund and H. J. Siegel, Heterogeneous processing. *Computer*, **26** (6): 13-17, 1993.

[GhY93] A. Ghafoor and J. Yang, Distributed heterogeneous supercomputing management system. *Computer*, **26** (6): 78-86, 1993.

[GrN97] A. S. Grimshaw, A. Nguyen-Tuong, M. J. Lewis, and M. Hyett, Campus-wide computing: Early results using Legion at the University of Virginia. *The Int. J. Supercomputer Applications and High Performance Computing*, **11** (2): 129-143, 1997.

[GrW94] A. S. Grimshaw, J. B. Weissman, E. A. West, and E. Loyot, Meta systems: An approach combining parallel processing and heterogeneous distributed computing systems. *J. Parallel and Distributed Computing*, **21** (3): 257-270, 1994.

[HaL95] B. Hamidzadeh, D. J. Lilja, and Y. Atif, Dynamic scheduling techniques for heterogeneous computing systems. *Concurrency: Practice and Experience*, **7** (7): 633-652, 1995.

[IvO95] M. A. Iverson, F. Ozguner, and G. J. Follen, Parallelizing existing applications in a distributed heterogeneous environment. *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 93-100.

[KaA97] M. Kafil and I. Ahmad, Optimal task assignment in heterogeneous computing systems. *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 135-146.

[KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, Heterogeneous computing: Challenges and opportunities. *Computer*, **26** (6): 18-27, 1993.

[KlM93] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, A case study in metacomputing: Distributed simulations of mixing in turbulent convection. *2nd Workshop on Heterogeneous Processing (WHP '93)*, Apr. 1993, pp. 101-106.

[LeP95] C. Leangsuksun, J. Potter, and S. Scott, Dynamic task mapping algorithms for a distributed heterogeneous computing environment. *4th Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30-34.

[MaB98] M. Maheswaran, T. D. Braun, and H. J. Siegel, High-performance mixed-machine heterogeneous computing, *6th Euromicro Workshop on Parallel and Distributed Processing,* Jan. 1998, pp. 3-9.

[MaS98] M. Maheswaran and H. J. Siegel, A dynamic matching and scheduling algorithm for heterogeneous computing systems. *7th Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 57-69.

[NaY94] B. Narahari, A. Youssef, and H. A. Choi, Matching and scheduling in a generalized optimal selection theory. *3rd Heterogeneous Computing Workshop (HCW '94)*, Apr. 1994, pp. 3-8.

[NoB96] M. L. Norman, P. Beckman, G. Bryan, J. Dubinski, D. Gannon, L. Hernquist, K. Keahey, J. P. Ostriker, J. Shalf, J. Welling and S. Yang, Galaxies collide on the I-way: An example of heterogeneous wide-area collaborative supercomputing. *The Int. J. of Supercomputer Applications and High Performance Computing*, **10** (2/3): 132-144, 1996.

[PeG91] D. Pease, A. Ghafoor, I. Ahmad, D. L. Andrews, K. Foudil-Bey, T. E. Karpinski, M. A. Mikki, and M. Zerrouki, PAWS: A performance evaluation tool for parallel computing systems. *Computer*, **24** (1): 18-29, 1991.

[SeS96] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima, Ninf: Network based information library for globally high performance computing. *Parallel Object-Oriented Methods and Applications (POOMA)*, 1996.

[ShW96] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, 98-117.

[SiA96] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, Heterogeneous computing. In A. Y. Zomaya (ed.), *Parallel and Distributed Computing Handbook*, New York, NY: McGraw-Hill, pp. 725-761, 1996.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio, Software support for heterogeneous computing. In A. B. Tucker, Jr. (ed.), *The Computer Science and Engineering Handbook*, Boca Raton, FL: CRC Press, pp. 1886-1909, 1997.

[SiM96] H. J. Siegel, M. Maheswaran, D. W. Watson, J. K. Antonio, and M. J. Atallah, Mixed-mode system heterogeneous computing. In M. M. Eshaghian (ed.), *Heterogeneous Computing*, Norwood, MA: Artech House, pp. 19-65, 1996.

[SiL93] G. C. Sih and E. A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, **4** (2): 175-187, 1993.

[SiY97] H. Singh and A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms. *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86-97.

[Sun90] V. S. Sunderam, PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, **2** (4): 315-339, 1990.

[WaK92] M. Wang, S.-D. Kim, M. A. Nichols, R. F. Freund, H. J. Siegel, and W. G. Nation, Augmenting the optimal selection theory for superconcurrency. *1st Workshop on Heterogeneous Processing (WHP '92)*, Mar. 1992, pp. 13-22.

[WaS97] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *J. Parallel and Distributed Computing*, **47** (1): 8-22, 1997.

[WaS93] D. W. Watson, H. J. Siegel, J. K. Antonio, M. A. Nichols, and M. J. Atallah, A framework for compile-time selection of parallel modes in a SIMD/SPMD heterogeneous environment. *2nd Workshop on Heterogeneous Processing (WHP) '93),* Apr. 1993, pp. 57-64.

[WeW94] C. C. Weems, G. E. Weaver, and S. G. Dropsho, Linguistic support for heterogeneous parallel processing: A survey and an approach. *3rd Heterogeneous Computing Workshop (HCW '94)*, Apr. 1994, pp. 81-88.

[YaA93] J. Yang, I. Ahmad, and A. Ghafoor, Estimation of execution times on heterogeneous supercomputer architecture. *1993 Int. Conf. Parallel Processing (ICPP '93)*, Vol I, Aug. 1993, pp. 219-225.

A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems

Muthucumaru Maheswaran and Howard Jay Siegel

Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285 USA
{maheswar, hj}@ecn.purdue.edu

This work was supported by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-97-M-0900 and N62271-98-M-0217.

Abstract

A heterogeneous computing system provides a variety of different machines, orchestrated to perform an application whose subtasks have diverse execution requirements. The subtasks must be assigned to machines (matching) and ordered for execution (scheduling) such that the overall application execution time is minimized. A new dynamic mapping (matching and scheduling) heuristic called the hybrid remapper is presented here. The hybrid remapper is based on a centralized policy and improves a statically obtained initial matching and scheduling by remapping to reduce the overall execution time. The remapping is non-preemptive and the execution of the hybrid remapper can be overlapped with the execution of the subtasks. During application execution, the hybrid remapper uses run-time values for the subtask completion times and machine availability times whenever possible. Therefore, the hybrid remapper bases its decisions on a mixture of run-time and expected values. The potential of the hybrid remapper to improve the performance of initial static mappings is demonstrated using simulation studies.

Keywords: dynamic scheduling, heterogeneous computing, list scheduling, mapping, matching, parallel processing, scheduling.

1. Introduction

Different portions of a computationally intensive application often require different types of computations. In general, a given machine architecture with its associated compiler, operating system, and programming environment does not satisfy the computational requirements of all portions of an application equally well. However, a heterogeneous computing (HC) environment that consists of a heterogeneous suite of machines and high-speed interconnections provides a variety of architectural capabilities, which can be orchestrated to perform an application that has diverse computational requirements [2, 10, 14, 15]. The performance criterion for HC used in this paper is to minimize the completion time, i.e., the overall execution time of the application on the machine suite.

One way to exploit an HC environment is to decompose an application task into subtasks, where each subtask is computationally well suited to a single machine architecture. Different subtasks may be best suited for different machines. The subtasks may have data dependencies among them, which could result in the need for inter-machine communications. Once the subtasks are obtained, each subtask is assigned to a machine (matching). The subtasks and inter-machine data transfers are ordered (scheduling) such that the overall completion time of the application is minimized. It is well known that such a matching and scheduling (mapping) problem is, in general, NP-complete [3]. Therefore, many heuristics have been developed to obtain near-optimal solutions to the mapping problem. The heuristics can be either static (matching and scheduling decisions are made prior to application execution) or dynamic (matching and scheduling decisions are made during application execution).

Most static mapping heuristics assume that accurate estimates are available for (a) subtask computation times on various machines and (b) inter-machine data transfer times. Often, it is difficult to accurately estimate the above parameters prior to application execution. Therefore, this paper proposes a new dynamic algorithm, called the time. It is called the hybrid remapper because it uses some results based on an initial static mapping in conjunction with information available only at execution time.

The hybrid remapper heuristics presented here are based on the list scheduling class of algorithms (e.g., [1, 9]). An initial, statically obtained mapping is provided as input to the hybrid remapper. If the initial mapping is not provided, it should be obtained before running the hybrid remapper by executing a static mapping algorithm such as the baseline [18], genetic-algorithm-based mapper [18], or Levelized Min Time [9].

The hybrid remapper executes in two phases. The first phase of the hybrid remapper is executed prior to application execution. The set of subtasks is partitioned into blocks such that the subtasks in a block do not have any data dependencies among them. However, the order among the blocks is determined by the data dependencies that are present among the subtasks of the entire application. The second phase of the hybrid remapper, executed during application run time, involves remapping the subtasks. The remapping of a subtask is performed in an overlapped fashion with the execution of other subtasks. As the execution of the application proceeds, run-time values for some subtask completion times and machine availability times can be obtained. The hybrid remapper attempts to improve the initial matching and scheduling by using the run-time information that becomes available during application execution and the information that was obtained prior to the execution of the application. Thus, hybrid remapper's decisions are based on a mixture of run-time and expected values.

This research is part of a DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort that includes NPS (Naval Postgraduate School), NRaD (a Naval Laboratory), Purdue, and USC (University of Southern California). It builds on SmartNêt, an operational scheduling framework and system for managing resources in a heterogeneous environment developed at NRaD [6]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

The organization of this paper is as follows. The matching and scheduling problem and the associated assumptions are defined in Section 2. Three variants of the hybrid remapper heuristics are described in Section 3. Section 4 examines the data obtained from the simulation studies conducted to evaluate the performance of the hybrid remapper heuristic. In Section 5, related work is discussed. Finally, Section 6 gives some future research directions.

2. Problem Definition

The following assumptions are made regarding the application. The application is decomposed into multiple subtasks and the data dependencies among them are known and are represented by a directed acyclic graph (DAG). That is, the nodes in the DAG represent the subtasks and the links represent the data dependencies. An estimate of the expected computation time of each subtask on each machine in the HC suite is known a priori. This assumption is typically made when conducting mapping research (e.g., [4, 7, 13, 16]). Finding the expected computation time is another research problem. Approaches based on analytical benchmarking and task profiling are surveyed in [14, 15]. Any loops and data conditionals are assumed to be contained inside a subtask.

It is assumed that the hybrid remapper is running on a dedicated workstation and all mapping decisions are centralized. Once a subtask is mapped onto a machine it is inserted into a local job queue on that particular machine. The execution of the subtask is managed by the job control environment of the local machine. The subtask executions are non-preemptive. All input data items of a subtask must be received before its execution can begin, and none of its output data items are available until its execution is completed. These assumptions make the matching and scheduling problem in HC systems more manageable. Nevertheless, solving the mapping problem with these assumptions is a significant step toward solving the more general problem.

An application task is decomposed into a set of subtasks $\underline{S}$, where s_i is the i-th subtask. Let the HC environment consist of a set of machines $\underline{M}$, where m_j be the j-th machine. The estimated expected computation time of subtask s_i on machine m_j is given by $e_{i,j}$. The earliest time at which machine m_j is available is given by $A[j]$, where $|A| = |M|$.

The data communication time between two machines has two components: a fixed message latency for the first byte to arrive and a per byte message transfer time. An $|M| \times |M|$ communication matrix is used to hold these values for the HC suite. Similar matrices are used by other researchers in HC (e.g., [7, 13, 16]).

To facilitate the discussion in Section 3, a hypothetical node called an exit node is defined for the DAG as follows. An exit node (subtask) is a node with 0 computation time that is appended to the DAG such that there is a 0 data transfer time communication link to this node from every node in the DAG that does not have an output edge. The critical path for a node in the DAG is defined as the longest path from the given node to the exit node.

3. The Hybrid Remapper Algorithm

3.1. Overview

The notion behind most dynamic mapping algorithms is that due to the dynamic nature of the mapping problem, it is not efficient to use a fixed mapping computed statically. Therefore, most dynamic mappers regularly either generate the mapping or refine an existing mapping at various times during task execution. That is, dynamic

Because the mapping is performed in real time, it is necessary to use a fast algorithm to avoid any machine idle times that occur from having to wait for the mapper to complete its execution. In the hybrid remapper algorithm presented here, the partial mapping problem is solved using a list-based scheduling algorithm.

In the following subsections, three variants of the hybrid remapper algorithm are described. The first phase, common for all three variants of the hybrid remapper, involves partitioning the subtasks into blocks and assigning ranks to each subtask (where the rank indicates the subtask's priority for being mapped, as defined below). The variants of the hybrid remapper differ in the second phase by the minimization criteria they use and by the way they order the subtasks examined by the partial mapping problem. One variant of the hybrid remapper attempts to minimize the expected partial completion time at each remapping step, and the others attempt to minimize the overall expected completion time. Two variants of the hybrid remapper order the subtasks at each remapping step using ranks computed at compile time, and the other using a parameter computed at run time.

3.2. Partitioning and Rank Assignment

This first phase uses the initial static mapping, expected subtask computation times, and expected data transfer times to preprocess the DAG that represents the application. Initially, the DAG is partitioned into B blocks numbered consecutively from 0 to $B-1$. The partitioning is done such that the subtasks within a block are independent, i.e., there are no data dependencies among the subtasks in a block. All subtasks that send data to a subtask s_j in block k must be in any of blocks 0 to $k-1$. Furthermore, for each subtask s_j in block k there exists at least one incident edge (data dependency) such that the source subtask is in block $k-1$, i.e., an incident edge from some s_i. The $(B-1)$-th block includes the subtasks without any successors and the 0-th block includes only those subtasks without any predecessors. The exit node is not included in any block in the DAG partitioning. The three blocks obtained using this partitioning algorithm for an example seven node DAG is shown in Figure 1(a).

Once the subtasks in the DAG are partitioned, each subtask is assigned a rank by examining the subtasks from block $B-1$ to block 0. The rank of each subtask in the $(B-1)$-th block is set to its expected computation time on the machine to which it was assigned by the initial static matching. Now consider the k-th block, $0 \le k < B-1$. Recall $e_{i,x}$ is the expected computation time of the subtask s_i on machine m_x. Let $c_{i,j}$ be the data transfer time for a descendent s_j of s_i to get all the relevant data items from s_i. The value of $c_{i,j}$ will be dependent on the machines assigned to subtasks s_i and s_j by the initial mapping, and the information in the communication matrix. Let iss(s_i) be the immediate successor set of subtask s_i such that there is an arc from s_i to each member of iss(s_i) in the DAG. In the equation below, each $e_{i,x}$ implies subtask s_i is assigned to machine m_x by the initial mapping. With these definitions, the rank of a subtask s_i is given by:

$$\text{rank}(s_i) = e_{i,x} + \max_{s_j \in \text{iss}(s_i)} (c_{i,j} + \text{rank}(s_j))$$

Figure 1(b) illustrates the rank assignment process for the subtask s_i. The rank of a subtask can be interpreted as the length of the critical path from the point the given subtask is located on the DAG to the exit node, i.e., the time until the end of the execution of all its descendents. Two variants of the hybrid remapper described here are based on the heuristic idea that by executing the subtasks with higher ranks as quickly as possible, the overall expected completion time for the application can be minimized.

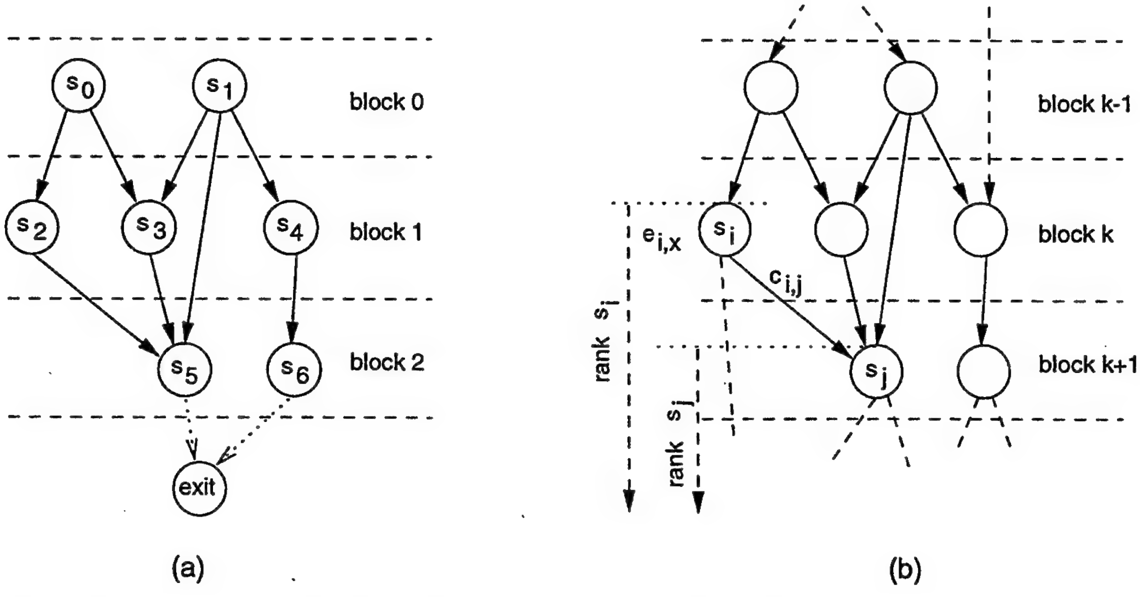


Figure 1: (a) Partitioning a DAG into blocks and (b) assigning ranks to the nodes of a DAG.

In all three variants of the hybrid remapper, the execution of the subtasks proceeds from block 0 to block $B-1$. A block k is considered to be executing if at least one subtask from block k is running. Also, the execution of several blocks can overlap with each other in time, i.e., subtasks from different blocks could be running at the same time.

The hybrid remapper changes the matching and scheduling of the subtasks in block k while the subtasks in block $(k-1)$ or before are being executed. The hybrid remapper starts examining the block k subtasks when the first block $(k-1)$ subtask begins its execution. When block k subtasks are being mapped, it is highly likely that run-time completion time information can be used for many subtasks from blocks 0 to $k-2$. There may be some subtasks from blocks 0 to $k-2$ that are still running or waiting execution when subtasks from block k are being considered for remapping. For such subtasks, expected completion times are used.

3.4. Minimum Partial Completion Time Static Priority (PS) Algorithm

As mentioned earlier, the hybrid remapper uses a list-scheduling type of algorithm to recompute the matching and scheduling for the subtasks in each block. In a list-scheduling type of algorithm, the subtasks are first ordered based on some priority. Then, each subtask is mapped to a machine by examining the list of subtasks from the highest priority subtask to the lowest priority subtask. The machine to which each subtask is assigned depends on the matching criterion used by the particular algorithm.

In this variant of the hybrid remapper, the priority of a subtask is equal to the rank of that subtask that was computed statically in the first phase (Subsection 3.2). The matching criterion used for subtask s_i is the minimization of the partial completion time, defined below. Thus, this variation is referred to as the minimum partial completion time static priority (PS) algorithm.

Let m_x be the machine on which s_i is being considered for execution. Then let $\mathrm{pct}(s_i,x)$ denote the partial completion time of the subtask s_i on machine m_x, $\mathrm{dr}(s_i)$ be the time at which the last data item required by s_i to begin its execution arrives at m_x, and $\mathrm{ips}(s_i)$ be the immediate predecessor set for subtask s_i such that there is an arc to s_i from each member of $\mathrm{ips}(s_i)$ in the DAG. For any subtask s_i in block 0, $\mathrm{pct}(s_i,x)=e_{i,x}$. For any subtask s_i not in block 0, where $s_j \in \mathrm{ips}(s_i)$, and s_j is currently mapped onto machine m_y,

$$\mathrm{dr}(s_i) = \max_{s_j \in \mathrm{ips}(s_i)} (c_{j,i} + \mathrm{pct}(s_j,y))$$
$$\mathrm{pct}(s_i,x) = e_{i,x} + \max(A[x], dr(s_i))$$

In the computation of $\mathrm{pct}(s_i,x)$, the above equation is recursively used until subtask s_j is such that its run-time completion time on machine m_y is available or subtask s_j is in block 0. The subtask s_i is remapped onto the machine m_x that gives the minimum $\mathrm{pct}(s_i,x)$, and $A[x]$ is updated using $\mathrm{pct}(s_i,x)$. Then the next subtask from the list is considered for remapping.

3.5. Minimum Completion Time Static Priority (CS) Algorithm

The notion behind the PS algorithm was that by remapping the highest rank subtask s_i to execute on the machine that will result in the smallest expected partial completion time, the overall completion time of the application may be minimized. Instead of this approach, the variant of the hybrid remapper described here attempts to minimize the overall completion time by remapping each subtask s_i in block k such that the length of the critical path through subtask s_i is reduced. Thus, this variation is referred to as the minimum completion time static priority (CS) algorithm. The reason for considering both PS and CS is that in PS the remapping is faster but CS attempts to derive a better mapping because it considers the whole critical path through s_i.

Let m_x be the machine on which s_i is being considered for execution. Then let the longest completion time path from a block 0 subtask to the exit node through the subtask s_i be $\mathrm{ct}(s_i,x)$. The overall completion time of the application task is determined by one such longest path through a block k subtask. Consider the subtask s_i in Figure 2. Assume that the longest path through s_i is shown by bold edges in Figure 2. For any subtask s_i,

$$\mathrm{ct}(s_i,x) = \max_{s_j \in \mathrm{iss}(s_i)} (\mathrm{pct}(s_i,x) + c_{i,j} + \mathrm{rank}(s_j)) \qquad = \mathrm{pct}(s_i,x) + \max_{s_j \in \mathrm{iss}(s_i)} (c_{i,j} + \mathrm{rank}(s_j))$$

The subtask s_i is remapped onto the machine m_x that gives the minimum $\mathrm{ct}(s_i,x)$, and $A[x]$ is updated using $\mathrm{pct}(s_i,x)$. Then the next subtask in the list is considered for remapping.

3.6. Minimum Completion Time Dynamic Priority (CD) Algorithm

The rank of a subtask s_i is computed prior to application execution. Therefore, if s_i is remapped to a machine other than the one it was assigned to by the initial static mapping, the rank of s_i may not give the length of the critical path from s_i to the exit node.

The algorithm presented here is same as the CS algorithm, except ranks are no longer used in ordering the subtasks within a block. Instead of using the statically computed ranks, this algorithm uses the value of $\mathrm{ct}(s_i,x)$, where m_x is the machine assigned to s_i in the initial mapping, to order the subtasks within a block. Thus, this within a block may not lead to the best overall completion time. In the given example, the DAG shown in Figure 3(a) is mapped onto two machines m_0 and m_1. Figure 3(b) shows the subtask computation time matrix, which gives the computation time of a subtask on different machines. The initial static mapping is shown in Figure 3(c). The numbers inside each bar correspond to the subtask index and the execution time of the subtask, in "subtask index/execution time" notation. The times are given in seconds. The data transfer times are negligible if the source and destination machines are the same, otherwise, for this example there is a fixed time of two seconds for the data transfer. In Figure 3(a), the number outside each node indicates the rank of that subtask derived using the initial mapping.

When block 2 is considered for remapping by either the PS or CS algorithm, s_5 is mapped first and then s_4 is mapped. Suppose s_0 finishes its execution in 20 seconds instead of 10 seconds and s_1 finishes in 10 seconds. This causes the subtask s_4 to become critical and s_5 to become non-critical, i.e., s_5 is not part of the critical path anymore. By using the rank numbers that were statically computed, the PS and CS algorithms map s_5 before s_4. Thus, s_5 will be mapped to the best machine and this can delay the completion of s_4. Instead of using the statically computed ranks, the CD algorithm considers ct(s_i,x), where subtask s_i is assigned to m_x in the initial mapping. For this example, subtask s_4 is assigned to machine m_0 and subtask s_5 is assigned to machine m_1. Therefore, the CD algorithm considers ct(s_4,0) and ct(s_5,1) to determine the remapping order.

ct(s_4,0) = 20+15+10+10 = 55
ct(s_5,1) = 10+20+10+2+10 = 52

Because the value of ct(s_5,1) is less than the value of ct(s_4,0), s_4 is considered for remapping before s_5 by the CD algorithm. This example illustrates that using ct(s_i, x), where m_x is the machine that is assigned to s_i in the initial mapping, enables the remapping algorithm to track the critical path better than using the static ranks.

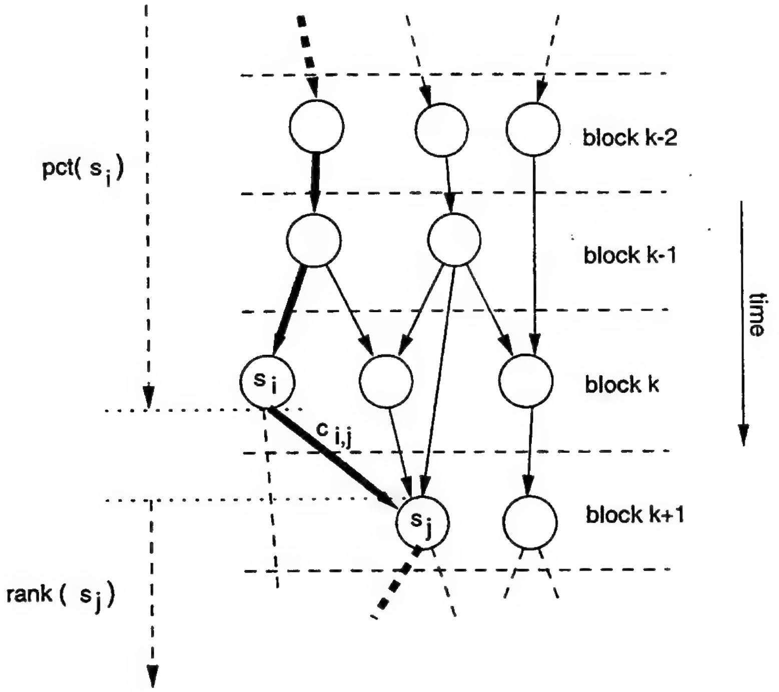


Figure 2: Estimating the completion time by considering the longest path through s_i.

4. Experimental Results and Discussion

4.1. Simulation Parameters

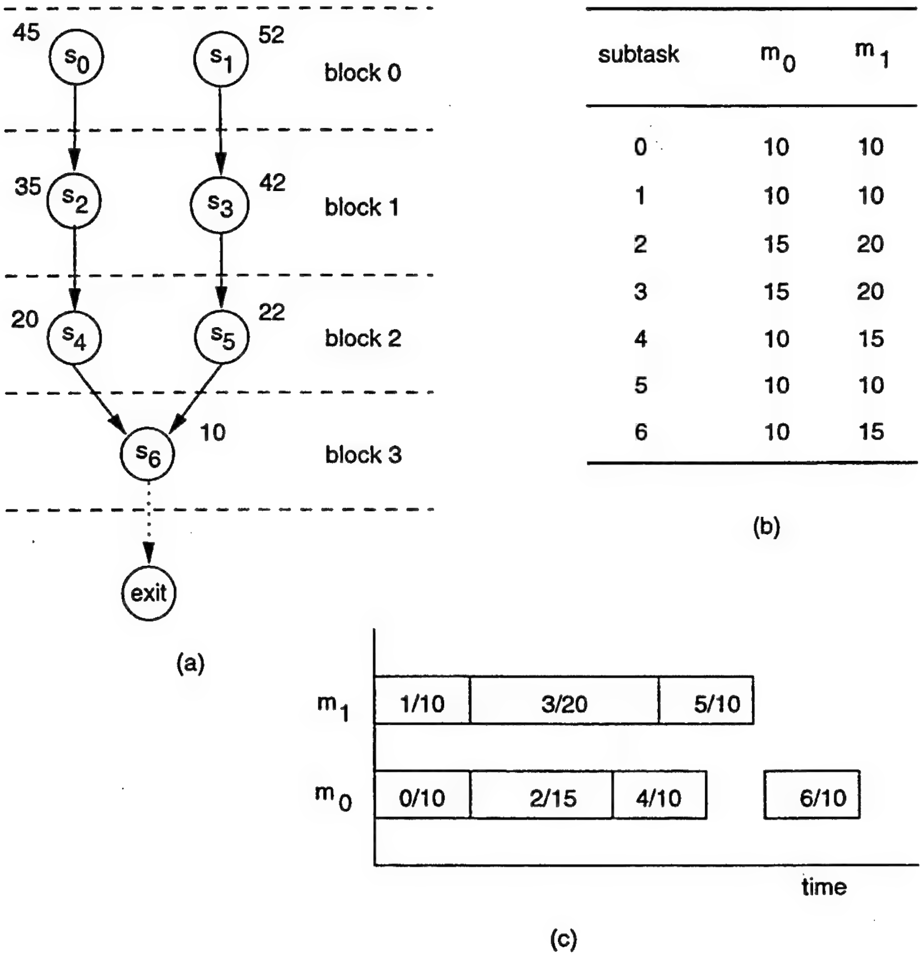
A simulator was implemented to evaluate the performance of the hybrid remapper variants. Various parameters are given as input to the simulator. Some parameters are specified as fixed values, e.g., number of machines, and others as a range of values with a maximum and a minimum value, e.g., subtask computation time. When a range is specified, the actual value is set to a random value within the specified range. Each data point in the results presented in this section is an average of 100 simulation runs. The experiments were performed on a Sun Ultra with a SPARC processor running at 165 MHz.

To generate a DAG that represents an application, the number of subtasks, maximum out degree of a node, number of data items to be transferred among different subtasks, range for subtask computation times, and range for data item sizes are provided as input to the simulator. Using these input parameters the simulator creates a table with the subtasks along the columns and data items along the rows. If a subtask s_i produces a data item d_i then the maximum out degree constraints are satisfied.

To define the HC suite, the number of machines is provided as input. The simulator randomly generates valid subtask computation times to fill a table that determines the subtask computation times on each machine in the HC suite. For these experiments it is assumed that a fully connected, contention-free communication network is used. The inter-machine communication times are source and destination dependent. Communication times are specified by a range value. The run-time value of a parameter such as the subtask execution time or inter-subtask data communication time can be different from the expected value of the parameter. The variation is modeled by generating simulated run-time values by sampling a probability distribution function (PDF) that has the expected value of the parameter as the mean.

4.2. Generational Scheduling

In this subsection, the generational scheduling (GS) algorithm [5] is briefly described. The performance of the hybrid remapper is compared with the performance of the GS algorithm in the next subsection. The GS algorithm is a dynamic mapping heuristic for HC systems.



subtask	m_0	m_1
0	10	10
1	10	10
2	15	20
3	15	20
4	10	15
5	10	10
6	10	15

(b)

Figure 3: An example mapping to illustrate the benefit of the CD algorithm: (a) the partitioned DAG, (b) the subtask computation time matrix, and (c) the initial mapping.

Initially, the GS forms a partial scheduling problem by pruning all the subtasks with unsatisfied precedence constraints from the initial DAG that represents the application. The initial partial scheduling problem consists of subtasks that correspond to those in block 0 of the hybrid remapper approach. The subtasks in the initial partial scheduling problem are then mapped onto the machines using an auxiliary scheduler. The auxiliary scheduler considers the subtasks for assignment in a first come first serve order. A subtask is assigned to a machine that minimizes the completion time of that particular subtask.

When a subtask from the initial partial scheduling problem completes its execution, the GS algorithm performs a remapping. During the remapping, the GS revises the partial scheduling problem by adding and removing subtasks from it. The completion of the subtask that triggered the remapping event may have satisfied the precedence constraints of some subtasks. These subtasks are added to the initial partial scheduling problem. The subtasks that have already started execution are removed from the initial partial scheduling problem. Once the revised partial scheduling problem is obtained, the subtasks in it are mapped onto the HC machine suite using the auxiliary scheduler. This procedure is iteratively performed until the completion of all subtasks.

From the discussions in Section 3, it can be noted that the hybrid remapper is provided with an initial mapping that is derived prior to application execution using a static matching and scheduling algorithm. The simulator generates a random DAG, using the parameters it receives as input, at the beginning of each simulation run. An initial static mapping for this DAG is obtained by matching and scheduling this DAG onto the HC suite using the baseline algorithm [18].

The baseline algorithm that is used to derive the initial mapping is a fast static matching and scheduling algorithm. It partitions the subtasks into blocks using an algorithm similar to the one described in Subsection 3.2. Once the subtasks are partitioned into blocks, they are ordered such that a subtask in block k comes before a subtask in block l, where $k<l$. The subtasks in the same block are arranged in descending order based on the number of descendents of each subtask (ties are broken arbitrarily). The subtasks are considered for assignment by traversing the list, beginning with block 0 subtasks. A subtask is assigned to the machine that gives the shortest time for that particular subtask to complete.

In this simulator, three different PDFs (a) Erlang(2) [12], (b) uniform, and (c) skewed uniform are used to generate the simulated run-time values. For Erlang(2), the expected values are provided as the mean and the PDF is sampled to obtain a simulated run-time value. In Figure 4, 10,000 consecutive random numbers generated by the Erlang(2) random number generator with mean ten is`shown using a 200-bin histogram. For the skewed uniform PDF, the following rule is used to generate the simulated run-time value. Let α_1 be the negative percentage deviation, α_2 be the positive percentage deviation, and u be a random number that is uniformly distributed in [0,1]. Then, the simulated run-time value of a parameter τ can be modeled as $\tau \times (100-\alpha_1+(\alpha_1+\alpha_2)u)/100$. For the uniform PDF, $\alpha_1 = \alpha_2 = \alpha$. For the simulation results presented here, Erlang(2) is used unless otherwise noted.

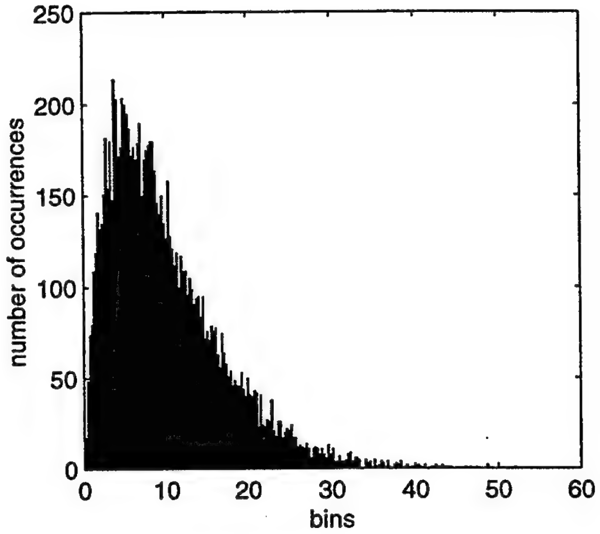
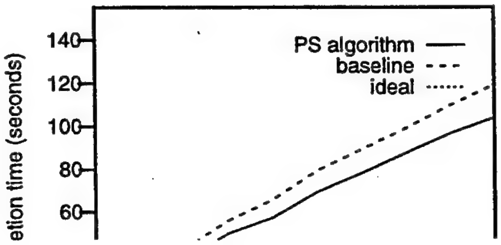
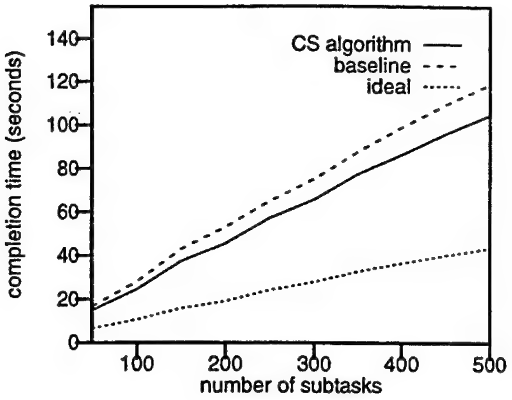


Figure 4: A 200-bin histogram for 10,000 consecutive samples of the Erlang(2) random number generator with mean equal to ten.

In these experiments, baseline refers to first deriving a static mapping using the baseline algorithm and expected subtask computation and communication times, and then, using this mapping, computing the total application execution time based on the simulated run-time values for computation and communication times. Also, in these experiments, ideal refers to deriving a static mapping using the baseline algorithm and simulated run-time values (instead of the expected values) for subtask computation and communication times. Note that this ideal is used for comparison purposes only, and cannot be implemented in practical environments. Also note that the ideal is not necessarily the optimal mapping. These simulated run-time values are also used to evaluate the application task completion time with the hybrid remapper variants.

In Figure 5(a), the performance of the PS algorithm is compared to the mapping that is obtained using the baseline algorithm for ten machines. Figure 5(b) shows a similar comparison for the CS algorithm for ten machines. The performance of the CD algorithm is shown in Figures 6(a) and 6(b). Figure 6(a) compares CD and the baseline for varying numbers of subtasks and ten machines. Figure 6(b) compares the two approaches for varying numbers of machines and 200 subtasks.





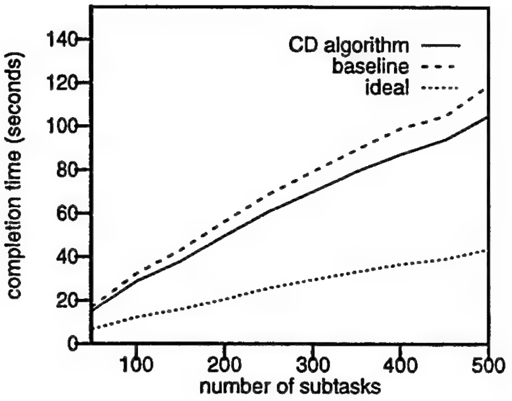
(b)

Figure 5: Performance of the hybrid remapper versus the baseline for (a) the PS algorithm and (b) the CS algorithm.

From Figures 5(a), 5(b), and 6(a) it can be observed that the performance difference among the three variants is almost negligible. The heuristic improvements performed to obtain the CS and CD variants from the PS variant of the hybrid remapper make the CS and CD use more initial matching and scheduling derived information. That is, while CS and CD use more information in an attempt to derive a better mapping than the PS, the information is based on expected values, rather than run-time values. Thus, there is no significant improvement. Also, in these simulation studies, the initial mapping is obtained using a simple baseline algorithm. The performance of CS and CD may improve if a higher quality initial assignment is used, e.g., if a genetic algorithm based mapper [18] is used for the initial matching and scheduling.

As the number of subtasks increase, the performance difference between each hybrid remapper variant and the baseline increases. This increase in performance can be attributed to two factors: (a) increased number of remapping events and (b) increased average number of subtasks per block. Increasing the number of remapping events provides the hybrid remapper with more opportunities to exploit the run-time values of parameters that are available during application execution. Also, with the increased number of subtasks per block the hybrid remapper can derive schedules that are very different from the initial schedule. Therefore, the average performance of the hybrid remapper increases with increasing number of subtasks.

Ten machines and 100 subtasks were used in Figure 7. In Figure 7(a), the performance of the CD algorithm is compared with the baseline for varying computation/communication ratios and Figure 7(b) shows the performance comparison of the CD algorithm with the baseline for varying average number of subtasks per block. Figure 7(a) shows that the hybrid remapper performs better as the computation/communication ratio increases. The computation/communication ratio is the average subtask execution time divided by the average inter-subtask communication time. In Figure 7(a), the low computation/communication ratio denotes the range 1.0-10.0, medium computation/communication ratio denotes the range 10.0-200.0, and high computation/communication ratio denotes the range 200.0-4000.0. With increasing computation/communication ratio, the data transfer times become less significant compared to the subtask computation times. The reason for the hybrid remapper not performing well with a low computation/communication ratio is currently under investigation.



(a)

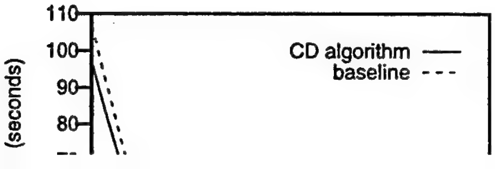
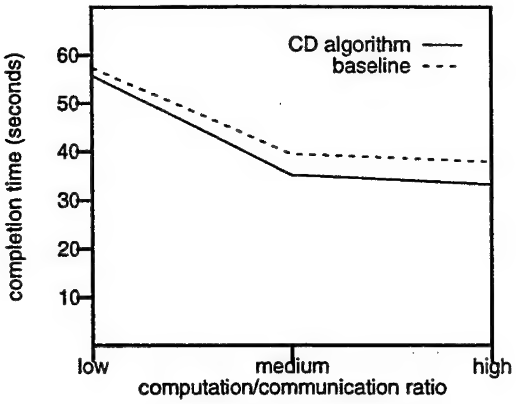
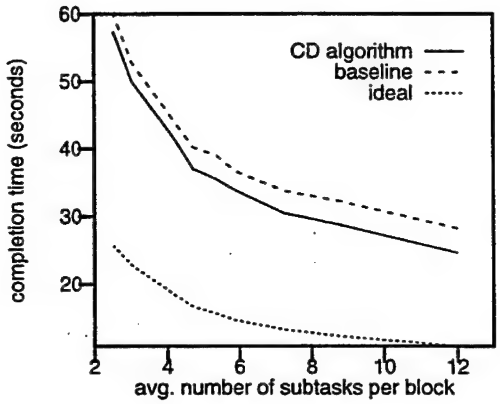


Figure 6: Performance of the CD algorithm versus the baseline for (a) varying the subtasks and (b) varying the machines.

From Figure 7(b) it can be noted that the relative performance of the CD algorithm increases with increasing the average number of subtasks per block. When there are more subtasks per block, it is possible for the hybrid remapper to derive mappings that are very different from the initial mapping.



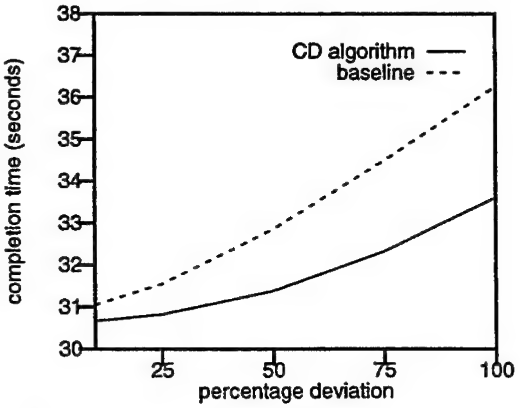
(a)



(b)

Figure 7: Performance of the CD algorithm versus the baseline for (a) varying the computation/communication ratio and (b) varying the average number of subtasks per block.

Figure 8(a) compares the performance of the CD algorithm with the baseline algorithm for a uniform distribution PDF, 20 machines, and 200 subtasks. Figure 8(b) performs the same comparison for a skewed uniform distribution PDF, 20 machines, and 200 subtasks. In the skewed uniform distribution the negative percentage deviation is half of the positive percentage deviation.



(a)

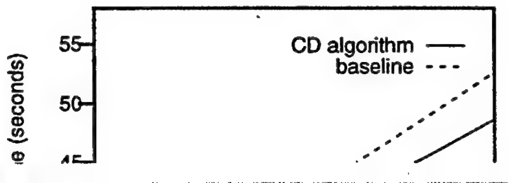
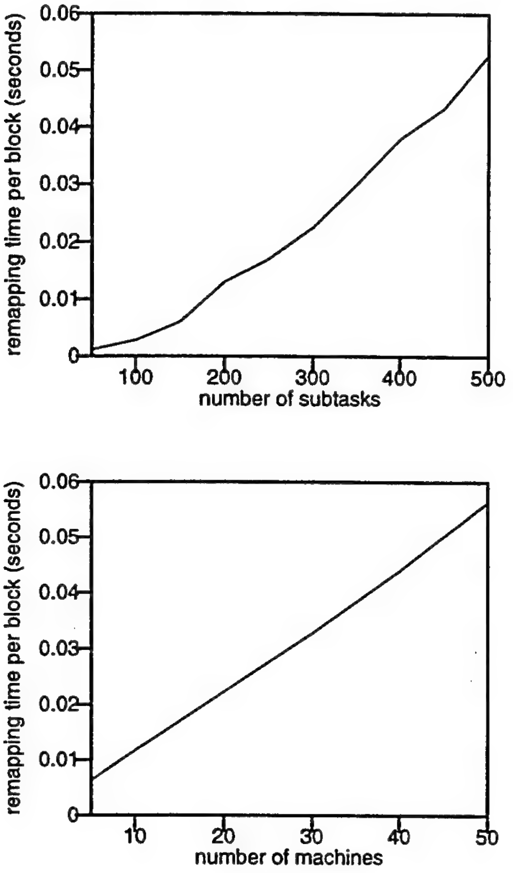


Figure 8: Performance of the CD algorithm versus the baseline for (a) using the uniform distribution for parameter modeling and (b) skewed uniform for parameter modeling.

As noted earlier, one of the features of the hybrid remapper algorithm that is presented here is overlapping its operation with the execution of the subtasks. To obtain complete overlap, in the worst case, the remapping time for a block of subtasks should be less than the execution time of the smallest subtask in the previous block. More precisely, the time available for remapping block k is equal to the difference between the time the first block $k-1$ subtask begins execution and the time the first block k subtask can begin execution. Figure 9(a) shows the per block remapping time for the CD algorithm for varying numbers of subtasks and ten machines. In Figure 9(b), the per block remapping time for the CD algorithm is shown for varying numbers of machines and 200 subtasks.



(a)

(b)

Figure 9: Per block remapping time of CD for (a) varying subtasks and (b) varying machines.

In Figure 10, the performance of the CD algorithm is compared with the GS algorithm for varying numbers of subtasks. From the simulation results it can be observed that the CD algorithm is slightly outperforming the GS algorithm. From the discussions in Section 3.6, it can be noted that the CD algorithm attempts to minimize the length of the critical path at each remapping step. In the GS algorithm, the critical path through the DAG is not considered when the subtasks are remapped. This is one reason for the better performance of the CD algorithm. The GS algorithm has more remapping events compared to the hybrid remapper. The number of remapping events is equal to the number of subtasks in the GS algorithm and equal to the number of blocks in the CD algorithm. The increased number of remappings allows the GS algorithm to base its assignment decisions on more current values. This may be why the GS is performing only three to four percent worst than the CD algorithm even though GS does not consider the critical path through the DAG. In the GS algorithm, at least one machine may be waiting on the scheduler to finish the mapping process. This scheduler induced wait time on the HC suite was not included in the GS versus CD comparison.

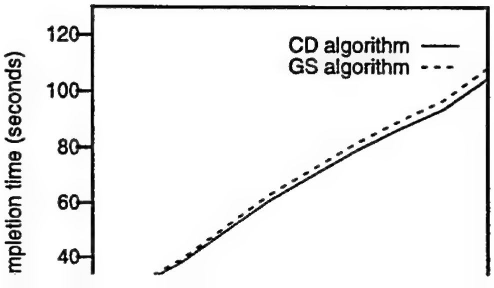


Figure 10: Performance of the CD algorithm versus the Generational Scheduling algorithm for varying numbers of subtasks.

5. Related Work

Other groups have also studied dynamic mapping heuristics for HC systems (e.g., [5, 8, 11]). A brief description of the GS algorithm and an experimental comparison of the hybrid remapper with the GS algorithm were presented in Section 4. The Self-Adjusting Scheduling for Heterogeneous Systems (SASH) algorithm is presented in [8]. One of the differences between the hybrid remapper and the SASH algorithm is that the hybrid remapper uses a list-scheduling based algorithm to perform the remappings at run time, whereas the SASH algorithm uses a variation of the branch and bound algorithm to generate the partial mappings at each remapping event. Also, unlike the GS and SASH algorithms, the hybrid remapper presented here can use any initial mapping to guide its remapping decisions, i.e., the initial mapping is used to compute the ranks and completion time estimates in the hybrid remapper. It is necessary to experimentally determine how the quality of the initial mapping impacts the overall performance of the hybrid remapper.

In [11], two mapping algorithms are presented. One is based on a distributed model and the other is based on a centralized model. The distributed mapping algorithm is different from the algorithms presented in [5, 8], and the hybrid remapper presented here, which are all centralized algorithms. The centralized mapping algorithm is based on a global queue equalization algorithm.

6. Conclusions and Future Work

The simulation results indicate that the performance of a statically obtained initial mapping can be improved by the hybrid remapper. From the simulation results obtained, performance improvement can be as much as 15% for some cases. The timings also indicate that the remapping time needed per block of subtasks is in the order hundreds of milliseconds for up to 50 machines and 500 subtasks. In the worst case situation, to obtain complete overlap, the computation time for the shortest running subtask must be greater than the per block remapping time.

The experimental studies revealed that the hybrid remapper performs better than the generational scheduling, but the margin of difference was only three to four percent. The hybrid remapper has a better machine utilization compared to the generational scheduling algorithm, because in the hybrid remapper the mapping operations are overlapped with the application execution. Further research is necessary to develop ways to improve the hybrid remapper's performance. This include examining the use of different schemes for partitioning the DAG into blocks, exploring the use of different ways of ordering subtasks within a block, and investigating the use of different criteria for determining subtask to machine assignments.

The partitioning scheme that is currently used in the hybrid remapper does not consider the usage pattern of the data items produced by a subtask. The partitioning is solely based on the data dependencies. This could cause a subtask with low rank value in a block k to be mapped before a subtask with high rank value in a block l, where $l > k$. Various alternate partitioning schemes need to be explored and evaluated to examine different criteria for forming blocks.

One of the features of the hybrid remapper algorithm presented here is the overlap of the execution of the hybrid remapper algorithm with the execution of the subtasks. In the hybrid remapper developed in this research, the remapping event for block k is the readiness to execute of the first block $k-1$ subtask. Hence, the number of remapping events is equal to the number of blocks. In other algorithms, such as the Generational Scheduling algorithm [5], the number of remapping events is equal to the number of subtasks. It is necessary to study the trade-offs of increasing the number of remapping events on the performance of the algorithms and the amount of machine idle time from having to wait for a mapping decision. Also, the interaction of varying the amount of uncertainty in the parameter values and increasing the number of remapping events needs further research.

In this paper, the performance of the hybrid remapper is compared with the performance of the static baseline, and the dynamic generational scheduling algorithm [5]. Further simulation studies are necessary to compare the performance of the hybrid remapper with other dynamic mapping algorithms, such as the queue equalization algorithm [11].

The hybrid remapper developed in this research assumed a fully connected, contention-free communication model. This model needs to be improved to accommodate message contention and restricted inter-machine network topologies that occur in practical situations. Also, enhancements are necessary to support cases where a subtask can have multiple sources (machines) for a needed data item [17].

The performance of the hybrid remapper has been studied using simulations in this research. Exploring the possibility of obtaining performance bounds using analytical methods is yet another possible area of future research.

Another future area of study is to evaluate the performance of the hybrid remapper when the initial mapping is generated by a genetic algorithm (GA) based mapper [18]. Also, it would be interesting to compare the relative performance of the hybrid remapper and the mapping obtained by a static GA-based mapper as the run-time values of the parameters deviate from their expected values.

In summary, a new dynamic mapping algorithm called the hybrid remapper was presented in this paper. The hybrid remapper uses novel heuristic approaches to dynamically improve a statically obtained initial mapping. The potential of the hybrid remapper to improve the performance of initial static mappings was demonstrated using

Acknowledgments -- The authors thank Robert Armstrong, Tracy Braun, Debra Hensgen, Taylor Kidd, Yu-Kwong Kwok, and Viktor Prasanna for their comments and useful discussions.

References

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Comm. of the ACM,* Vol. 17, No. 12, Dec. 1974, pp. 685-690.

[2] M. M. Eshaghian, ed., *Heterogeneous Computing,* Artech House, Norwood, MA, 1996.

[3] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Trans. on Software Engineering,* Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.

[4] R. F. Freund, "The challenges of heterogeneous computing," *Parallel Systems Fair at the 8th Int'l Parallel Processing Symp.,* Apr. 1994, pp. 84-91.

[5] R. F. Freund, B. R. Carter, D. Watson, E. Keith, and F. Mirabile, "Generational scheduling for heterogeneous computing systems," *Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '96),* Aug. 1996, pp. 769-778.

[6] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for meta-computing," *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96),* June 1996, pp. 514-521.

[7] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer,* Vol. 26, No. 6, June 1993, pp. 78-86.

[8] B. Hamidzadeh, D. J. Lilja, and Y. Atif, "Dynamic scheduling techniques for heterogeneous computing systems," *Concurrency: Practice and Experience,* Vol. 7, No. 7, Oct. 1995, pp. 633-652.

[9] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 93-100.

[10] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer,* Vol. 26, No. 6, June 1993, pp. 18-27.

[11] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 30-34.

[12] A. Papoulis, *Probability, Random Variables, and Stochastic Processes, Second Edition,* McGraw-Hill, New York, NY, 1984.

[13] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th Heterogeneous Computing Workshop (HCW '96),* Apr. 1996, pp. 98-117.

[14] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook,* A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[15] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook,* A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[16] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW '96),* Apr. 1996, pp. 86-97.

[17] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE Trans. on Parallel and Distributed Systems,* Vol. 8, No. 8, Aug. 1997, pp. 857-871.

[18] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. of Parallel and Distributed Computing,* Special Issue on Parallel Evolutionary Computing, accepted and scheduled to appear.

Biographies

Muthucumaru Maheswaran is a PhD candidate in the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana, USA. His research interests include task matching and scheduling in heterogeneous computing environments, parallel languages, data parallel algorithms, distributed computing, scientific computation, and world wide web systems. He has authored or coauthored two journal papers, seven conference papers, two book chapters, and one technical report.

Mr. Maheswaran received a BScEng degree in electrical engineering from the University of Peradeniya, Sri Lanka, in 1990 and a MSEE degree from Purdue University in 1994. Mr. Maheswaran is a member of the Eta Kappu Nu honorary society.

Howard Jay Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE (1990) and a Fellow of the ACM (1998). He received BS degrees in both electrical engineering and management (1972) from MIT, and the MA (1974), MSE (1974), and PhD degrees (1977) from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing* (second edition 1990). He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing* (1989-1991), and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* (1993-1996) and the *IEEE Transactions on Computers* (1993-1996). He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Prof. Siegel's heterogeneous computing research includes modeling, mapping heuristics, and minimization of inter-machine communication. He is an Investigator on the Management System for Heterogeneous Networks (MSHN) project, supported by the DARPA/ITO Quorum program to create a management system for a heterogeneous network of machines. He is the Principal Investigator of a joint ONR-DARPA/ISO grant to design efficient methodologies for communication in the heterogeneous environment of the Battlefield Awareness and Data reconfigurable parallel machine. His algorithm work involves minimizing execution time by exploiting architectural features of parallel machines. Topological properties and fault tolerance are the focus of his research on interconnection networks for parallel machines. He is investigating the utility of the dynamic reconfigurability and mixed-mode parallelism supported by the PASM design ideas and the small-scale prototype.

NPS-CS-99-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Approved for public release; distribution is unlimited.
Prepared for: Naval Postgraduate School

A Comparison of C++ Sockets and Corba in a Distributed Matrix Multiply Application

by

M. Schnaidt
A. Duman
T. Lewis

June 2, 1998

REPORT DOCUMENTATION PAGE

Form approved

OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
2 JUNE 98

3. REPORT TYPE AND DATES COVERED
TECHNICAL REPORT

4. TITLE AND SUBTITLE
A COMPARISON OF C++ SOCKETS AND CORBA IN A DISTRIBUTED MATRIX MULTIPLY APPLICATION

5. FUNDING

6. AUTHOR(S)
MATT SCHNAIDT, ALPAY DUMAN, and TED LEWIS

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School, 833 Dyer Road, Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-CS-99-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words.)

This project has two primary purposes. The first, is to implement a distributed matrix multiply algorithm using C++ sockets, anCorba objects with the objective of discovering what additional overhead, if any, exists in aCorba implementation. Secondly, attempt to improve the speedup through the use of stateful servers in the C++ implementation.

14. SUBJECT TERMS

Buffer Deadlock, Buffer Size Limitations, Client-Server Architecture, C++ Sockets, CORBA, Unix Name Service, Speedup, Efficiency, Superscalar Efficiency.

15. NUMBER OF PAGES 24

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassifed	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT**

Enclosure (6)

Technical Report # NPSCS-98-001 Alpay Duman and Matt Schnaidt

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

RADM Robert C. Chaplin
Superintendent

R. Elster
Provost

This report was prepared for Naval Postgraduate School.

This report was prepared by:
Matt Schnaidt and Alpay Duman.

Matt Schnaidt
Captain US Army

Alpay Duman
Lieutenant Jr. Grade Turkish Navy

Ted Lewis
Professor of Computer Science

Reviewed by:

Released by:

Dean D. Boger, Chair
Department of Computer Science

D. W.Netzer
Associate Provost and
Dean of Research

A Comparison of C++ Sockets and Corba in a Distributed Matrix Multiply Application

2 June 1998

CPT Matt Schnaidt, USA, mcschnai@cs.nps.navy.mil

LTJG Alpay Duman, TN, aduman@cs.nps.navy.mil

ABSTRACT: This project has two primary purposes. The first, is to implement a distributed matrix multiply algorithm using C++ sockets, and Corba objects with the objective of discovering what additional overhead, if any, exists in a Corba implementation. Secondly, attempt to improve the speedup through the use of stateful servers in the C++ implementation.

CONCEPTS:

Buffer Deadlock
Buffer Size Limitations
Client-Server Architecture
C++ Sockets
CORBA
Unix Name Service
CORBA Name Service
Speedup
Efficiency
Superscalar Efficiency

SUMMARY

Matt Schnaidt did the C++ Socket implementation of the matrix multiply. This included writing all of the C++ code (a 2d Array Class, the client and server code for the matrix multiply, and a name server), testing, debugging, and making record runs. I did this in three phases. In phase 1, I implemented a simple matrix multiply using UDP sockets; I used this to debug the multiply algorithm, and the interaction with the name server (which I called memberServer).

In phase 2, I converted the client and server to using TCP sockets so that they could reliably transmit messages which exceeded the maximum packet size without concern for ordering, lost or duplicate packets. I finished phase 2 by measuring the time it took to do matrix multiplies with a varied number of servers and varied size matrices.

In phase 3, I changed the server so that it remembers it state to cut down on message traffic time. I finished phase 3 by measuring the time it took to matrix multiplies, again with a varied number of servers and varied size matrices.

Alpay and I divided the report writing, and slide development for the presentation equally.

Alpay Duman did the CORBA implementation of the matrix multiply. This included writing the IDL, the client code and the interface object implementation defined in the IDL file, testing, debugging and making record runs. I did this in three phases.

In phase one, I implemented the IDL file and the object implementation for this definition. For passing the array I used type sequence in Interface Definition Language, which is a linear dynamic container.

In phase two, I implemented the client invoking the object implementation by using deferred synchronous method, which is a non-blocking dynamic remote procedure call method.

In phase three, I used a special function for CORBA send_multiple_requests_deferred(), which initiates a number of requests in parallel. It does not wait for the requests to finish before returning to the caller.

Table of Contents

1. **Problem Statement**. Develop a distributed matrix multiply algorithm, using C++ sockets and Corba. Compare the performance of the two algorithms in order to discover the overhead associated with the use of Corba.

   Given 2 matrices, A and B, multiply them together to produce a result matrix C. The dimensions of the matrices areA(i X j), B(j X k), C(i X k). Note that the number of columns in A must equal the number of rows in C.

   The general algorithm for solving this problem is:

```
for(int ix = 0; ix <rowInA; ix++){
  for(int jx = 0; jx < j; j++){

    C[i, j] = 0;
    for(int kx = 0; kx < k; k++){
      C[i, j] += A[i, k] * B[k,j];
    }//end for
  }//end for
}//end for
```

2. **Approach**.

A. Distributed Matrix Multiply Algorithm. We agreed, prior to any implementation, on a generic algorithm that we would each implement. One of our primary concerns was that both implementations time the same, or equivalent, events so that we could do a meaningful comparison. Below are listed the steps of our generic algorithm.

- Get the dimensions of the A and B matrices.
- Dynamically allocate space for the A and B matrices.
- Get the filename for the A matrix. Open the file, load the matrix into memory, and close the file.
- Get the filename for the B matrix. Open the file, load the matrix into memory, and close the file.
- Get the number of servers to use.
- START THE TIMER (using gettimeofday() system call).
- Access the name server (must do this explicitly in C++).
- Do the matrix multiply. Send one row fromA, and the entire B matrix to every server, until every row has been distributed.
- Receive the result row of C from each server. Assemble the result matrix.
- STOP THE TIMER.
- Display or write to disk the result matrix.
- Calculate and display the elapsed time.

B. C++ Sockets Implementation.

In the C++ implementation, there is one memberServer that listens at a fixed port and host (declared as MEMBER_SERVER_PORT and MEMBER_SERVER_HOST in file memberServer.h).

Each matrixClient and matrixServer uses three sockets. One is a udp socket to communicate with the memberServer. The other two are tcp sockets; one is a "server" socket at which it listens to receive messages, and the other is a "client" socket that it uses to send messages (see Figure 1, Figure 2, and example at the end of this section).

There can be many matrixServers, but only one permachine as each listens at a fixed port (declared as MATRIX_SERVER_PORT in memberServer.h).

There can be many matrixClients when using the stateless matrixServers, but only one at a time when using the stateful matrixServer (this will be further explained in section 3b). Each matrixClient listens at a fixed port number (declared as CLIENT_RCV_PORT in memberServer.h).

When a matrixServer starts up, it registers with the memberServer. The memberServer keeps a list of the ip address of every matrixServer that has registered with it. The matrixClient then contacts the matrixServer to get the ip addresses of every matrixServer.

The matrixClient uses these ip addresses together with MATRIX_SERVER_PORT to send the following workRequest to the matrixServer:
Index of result into C matrix, number of columns in A, number of columns in B, a row from matrix A, B matrix.

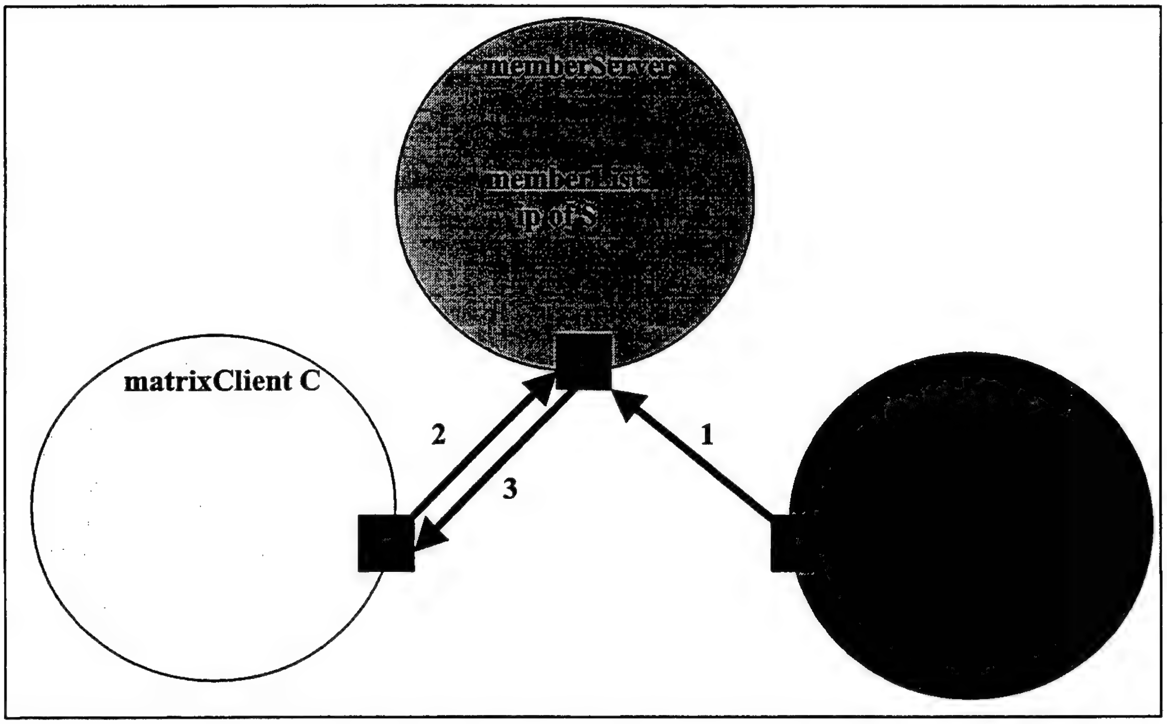


Figure 1: Interaction with memberServer

(1. MatrixServer S registers with memberServer. memberServer records ip of server. 2. matrixClient queries member Server for all matrixServers. 3. memberServer sends ip address of matrixServer S.)

To demonstrate how the distributed algorithm works, consider this examplescenario(refer to figure 1 for steps 1 – 6, figure2 for steps 7 – 9):

1) memberServer starts up, listens at udp socket.
2) matrixServer S starts up, registers with memberServer, and listens at its server tcp socket.
3) memberServer adds matrixServer S's ip address to memberList. Steps 2 and 3 are repeated for matrixServer T and U.

4) matrixClient starts up, prompts user for matrix sizes and filenames, loads matrices.
5) matrixClient asks user number of servers to use.
6) matrixClient contacts memberServer. memberServer responds with all ip addresses of matrixServers; memberServer puts this information into a list.
7) matrixClient forks off a child process which will cycle through the list, up to the max number of servers (input by user), connecting frommatrixClient's client tcp socket and sending work request (format given in preceding paragraph) to each matrixServer. Meanwhile, main process closes the client tcp socket, and listens at matrixClient's server tcp socket for replies.
8) matrixServers receive work requests from matrixClient on client tcp sockets. Each uses number of columns in A and B to allocate space for the A matrix row and the B matrix, calculates result row C, and connects with matrixClient from matrixServer's client port and sends result rows and indexes of result row to matrixClient.
9) matrixClient receives all result rows, and fills the C matrixmatrixClient's child process exits when all requests sent; parent process stops timer once C matrix is fully constructed.

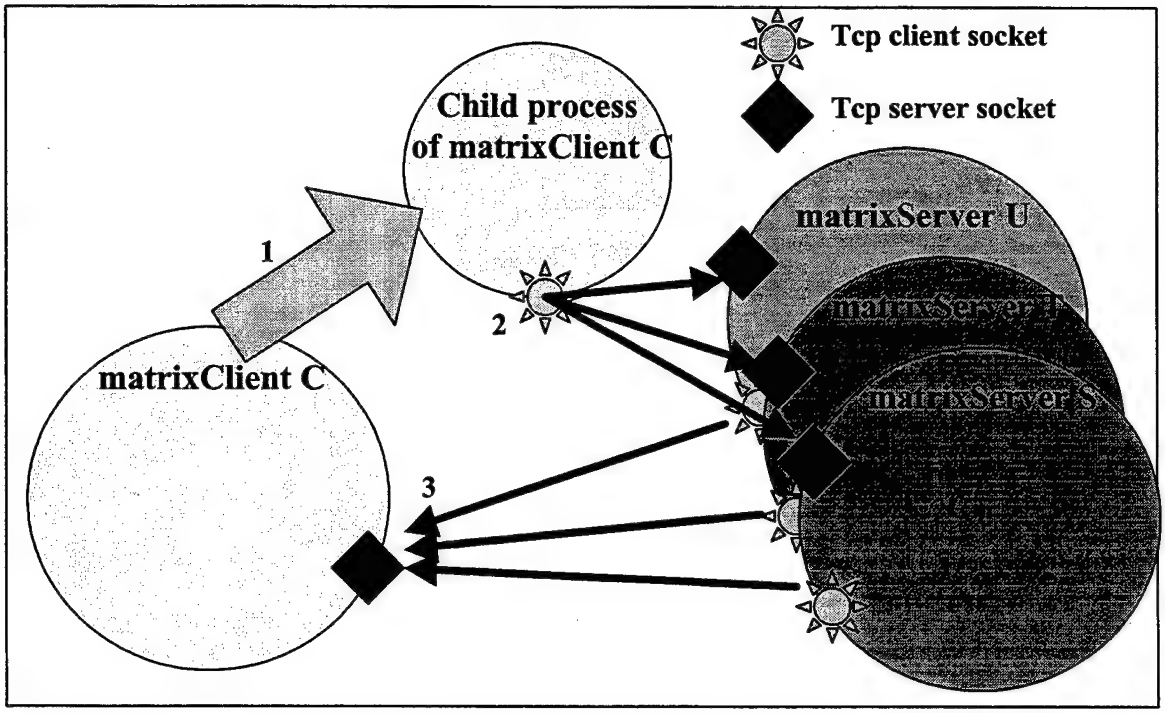


Figure 2: Distributed Matrix Multiply

(1. matrixClient forks child process. 2. Child sends all requests for work to servers. 3. matrixServers reply to the main process.

Note that this implementation of the matrixServer is stateless; that is, every time the matrixClient sends a request for work to the matrixServer, it must send the complete B matrix.

The advantage of this implementation is that requests for work to a single matrixServer may be interleaved from several matrixClients. Since the matrixServer has no state, it does not care from which client it received the current request; previous results have no impact on current calculations.

However, message sizes can be very large for every request to a server since we include both the row from the A matrix as well as the entire B matrix – even if we are sending a subsequent row computation to the same server. In order to speed up the calculations, we implemented a stateful matrixServer. On the first request sent to each matrixServer, the matrixClient sends the full message: index, #cols in A, # cols in B, A row, B matrix. On subsequent sends to the servers, the matrixClient setscols in B equal to zero and sends no B matrix. On the matrixServer side, if cols in B is nonzero, the server deletes the old B matrix, allocates space for and receives the B matrix. But ifcols in B is zero, the server uses the B matrix it currently has in memory. We found that this drastically improved speedup. Again, the drawback is that a stateful matrixServer cannot gracefully handle interleaved requests.

A work around for the interleaved work request problem, that we did not implement, is to send some identification to the matrixServer along with the work requests (eg the ip of the client concatenated with the process id of the current process). Once a matrix server begins to serve a client, it will only accept requests for service from that client until the client sends a termination, or end of job, flag. It would send a request denied message to any other processes requesting service until its current client released it with the end of job flag. This would prevent errors from occurring due to interleaved calculations of unrelated matrix multiplies, but may not be the best use of resources as each server would be dedicated to one client until the client released it. Also, if the end of job flag was not received by the matrix server (if the client crashes, or the message is lost), the server could wait forever and be unavailable to all other processes.

C. Corba Implementation

In CORBA implementation we already had the Naming Service available for us. We used OrbixNames as Naming Service, which had a Load Balancing feature with round robin scheduling. The Naming Service was respondingto the clients with an object reference of an available server at the head of the queue.

Each host machine, where client, Naming Service and servers were residing, had an Orbix Daemon running. The communication between client and servers were handled by the daemons, as well-known contact points.

When an object implementations server starts up, it registers its object implementation to the naming server and reports that it is ready to receive calls, where it is added to the list of available servers for a specific group.(i.e. MULTgrp the available object implementations for matrix multiplication.)

When the client contacts the Naming Service, ituses the name MULTgrp to get the object reference for the first available server in the queue for the group, because round robin algorithm is used for scheduling. In this case this causes a Load Balancing throughout the object implementations just like a SPMD machine.

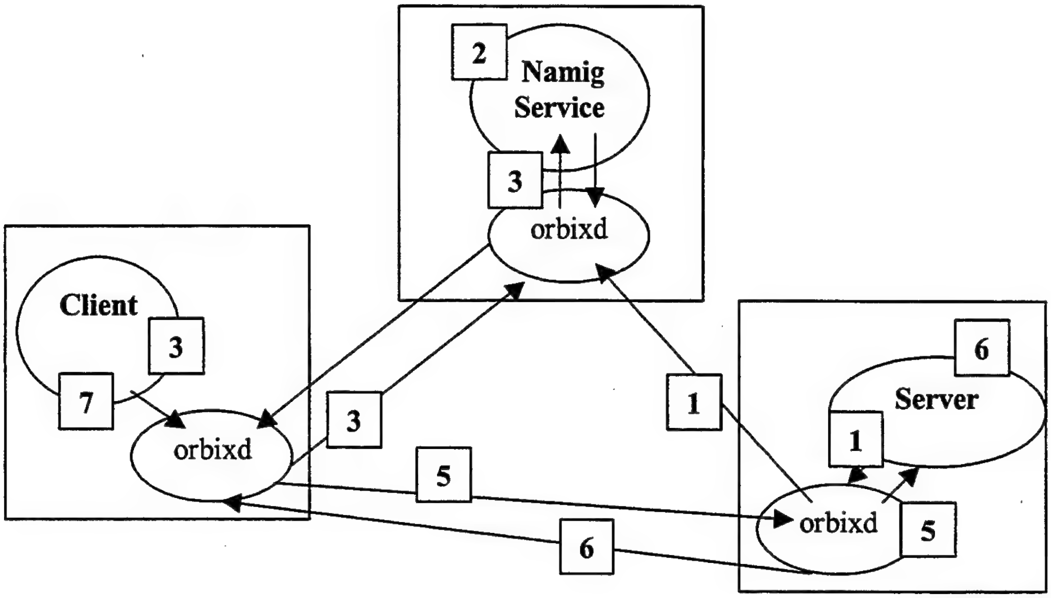


Figure 3: Distributed Matrix Multiply

To demonstrate how the distributed algorithm works, consider this examplescenario(refer to Figure 3)

1. Object implementations server starts up, and registers the object implementation to the naming server.(through local daemon)
2. The Naming service adds the ref to the end of the queue of that group.(MULTgrp)
3. Client starts up, prompts user for matrix sizes and filenames,loads matrices. Client tries to resolve the name MULTgrp through Naming Service. Naming Service looks up for that group and returns the object reference at the head of the queue.
4. Client gets a reference to Naming Service and resolves nameMULTgrp to get a reference for the first available object implementation.
5. Client creates the request objects and populates them with a row and the second matrix. Finally it invokessend_multiple_requests_deferred() on the orbixd and sends all the requests parallel. Starts waiting for the results. Itpoll() the request objects to collect the results.
6. Servers receive work requests from client through daemons. They compute the result for each row and return them.
7. Client collects the results available in the request objects and constructs the result matrix.

3. Results.

A. Corba versus C++ Sockets.

The comparison experiment consisted of 5 trials each of 10 X 10, 100 X 100 and 200 X 200 matrix multiplies on: 1) no servers (establishes a base), 2) 1 server on same machine, 3) 1 server on different machine, 4) 5 servers on different machines, and 5) 10 servers on different machines from client. The machines were 300 MHz Sun Ultra 10's connected via 10Mbitethernet in Spanagel 506; the tests were done at times when there were no other users using the machines. The machines used are listed in Appendix 6.d.

Upon receipt of the results, we found that the time measurements of the 10 X 10 matrix multiply were too variable to provide useful insight, so we focused on the 100 X 100 and 200 X 200 matrices. By referring toTable 1, we see that if we exclude the 1st result row, the use of CORBA adds between 23 and 41% to the completion time that you would expect using C++ sockets. If we restrict our review to the larger, and less variable, 200 X 200 matrix multiply we see that the range of added overhead is further restricted to between 26 and 29%.

The exception to our general observation occurs when using a server on the same machine as the client. In this situation, Corba takes almost as long as it would take to use a server on another machine. Contrast this to the C++ socket implementation that takes slightly longer than the baseline test of the non-distributed algorithm. This is because, even though the client and server reside on the same host, the ORB must repeatedly open and close a socket to itself for each request for service.

Comparison Between C++ Sockets and CORBA Matrix Multiply						
	C++ Sockets		CORBA		Socket/CORBA	
server configuration	100X100	200X200	100X100	200X200	100X100	200X200
no servers	0.2141	1.7874	0.0000	0.0000	N/A	N/A
1 server, same host	0.5571	3.2124	3.2891	29.6662	0.1694	0.1083
1 server, different host	3.6977	29.0047	4.5242	36.4906	0.8173	0.7949
5 servers, different host	3.7413	28.4360	4.5755	36.3644	0.8177	0.7820
10 servers, different host	3.7202	28.5166	5.2512	36.6698	0.7085	0.7777

Table 1: C++ Socket vs. Corba Matrix Multiply Performance

The detailed tables of results are in appendix 6A. Figure 4 below further depicts the overhead of Corba vs C++ sockets.

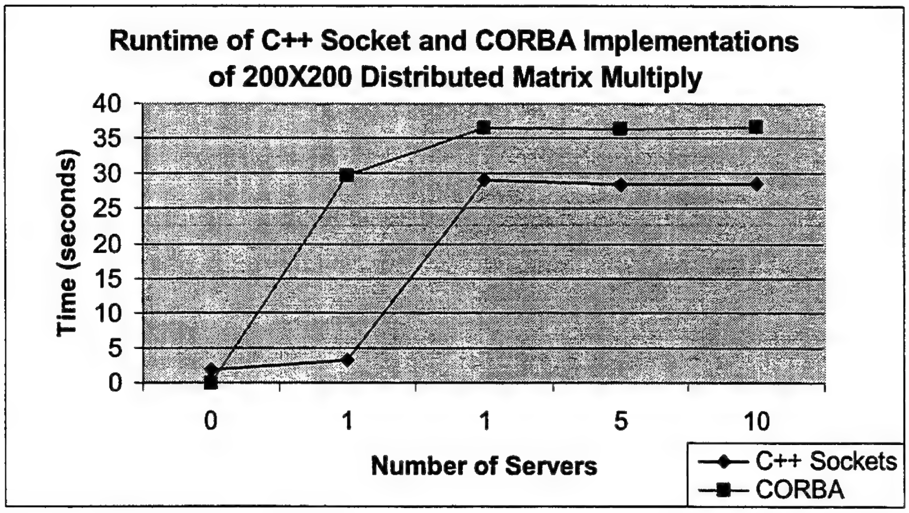


Figure 4: Socket vs Corba Matrix Multiply Performance

(Note: On the X axis, the first 1 represents client and server on same host, the second 1 represents client and server on different hosts).

We were going to end our analysis of Sockets vs Corba here, but then we realized that there were several other areas that were relevant to the analysis. These are training, development time, and code size.

The training required to learn C++ sockets, for someone who is trained in C++ and basic Unix, is approximately 1 week. Contrast this to the requirements for Corba proficiency that may be as great as 3 months.

The advantage of Corba becomes apparent in development time. While it took Matt about 1 week to write and fully debug his implementation, Alpay finished the Corba implementation in 1 day. This is significant when you consider the cost of a developer, the cost of equipment and facilities, and the cost of getting a product fielded or to market earlier than the competition.

Another advantage of Corba is the size of the code generated. Alpay's Corba implementation took 197 lines of code, while Matt's socket implementation took 859 lines. This is significant not only because of the costs of development, but from a maintenance standpoint. As lines of code increase, the potential for bugs also increases, as does the cost to repair those bugs. Additionally, shorter code is easier to understand and maintain.

So, when deciding whether to use Corba versus a lower level implementation, at least the factors addressed above must be considered.

B. C++ Sockets with Stateful Servers.

We were somewhat surprised to find no speedup in our initial implementation of the distributed algorithm. We realized that communication time greatly exceeded computation time because we were using loosely coupled servers implemented over TCP-IP. To try to observe speedup, we increased the problem size: we tried both 500 X 500 and 1000 X 1000 matrices, with no observed speedup. In fact, the distributed algorithm ran several times longer than the non-distributed algorithm.

We then did a quick and dirty big "O" analysis (appendix 6.E), and found that, not only did computation time increase with the cube of the problem size,but so did our message traffic and the corresponding messaging times. Because, as problem size increased, both computation and messaging complexity increased with$O(N^3)$, we realized that our current implementation of the matrix multiply algorithm would never show speed up.

Our solution was to implement stateful servers, as described at the end of section 2B. By using stateful servers, we only had to send the B matrix to each server once, then on every other transmission, we only sent each server the A row, index and dimensions. This reduced communication complexity from$O(N^3)$ to $O(N^2)$ (See Big O analysis, appendix 6.E). We found that if the problem size was large enough to overcome connection and message costs, we were able to see significant speed up results.

We found that the problem size had to be greater than 100 X 100 to see any speed up; at that problem size or lower, adding servers actually dropped speedup to less than 1 due to the cost of establishing connections and sending messages. This drop in speedup due to distributing too small of a problem is illustrated in Figure 5.

Note that all of our speedup calculations use the non-distributed (eg no server) results as the base for calculation.

The best speedup for a 500 X 500 matrix multiply occurred with 6 servers and was2.94; this speedup was achieved at an efficiency of0.490. The most efficient speedup was 0.9795 at an efficiency of 0.9795 using 1 server on a different host than the client These results are illustrated in Figure 6 and Figure 7 below.

The best speedup for a 1000 X 1000 matrix multiply occurred with 9 servers and was4.874; this speedup was achieved at an efficiency of0.542. The most efficient speedup was 1.032 at an efficiency of 1.032. This was achieved using 1 server on the same host as the client. This result shows superscalar speedup – the parallel version runs faster than the sequential version. We believe that, although we attempted to keep both the sequential and parallel algorithms as close as possible, the distributed algorithm introduced some more efficient methods than the sequential, causing thissuperscalar speedup. These results are illustrated inFigure and Figure 7 below.

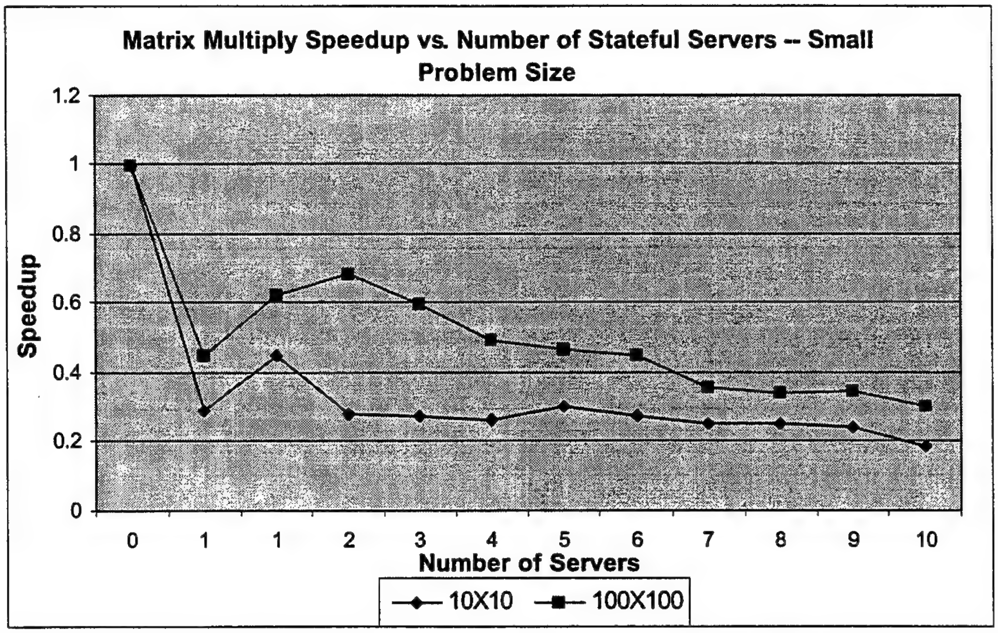


Figure 5: Speedup Curves – Small Problem Size

(Note: On the X axis, the first 1 represents client and server on same host, the second 1 represents client and server on different hosts).

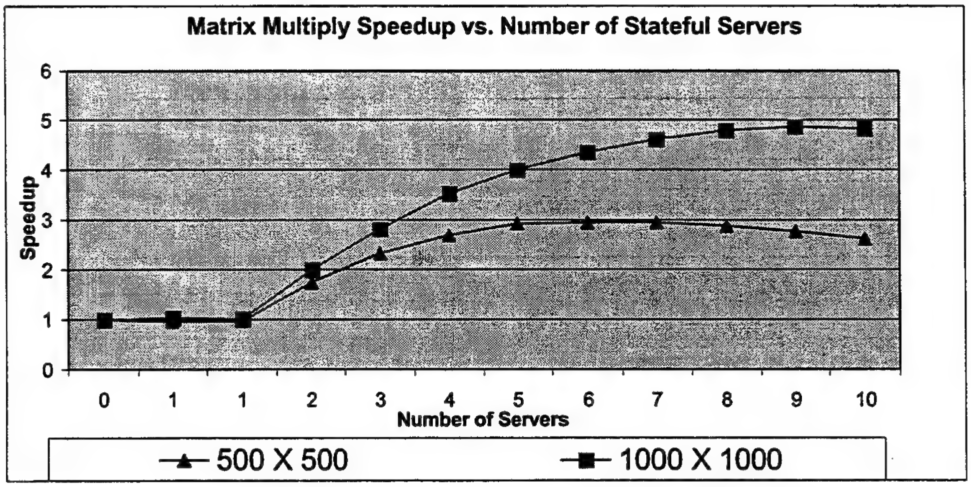


Figure 6: Stateful Server Speedup Curves

(Note: On the X axis, the first 1 represents client and server on same host, the second 1 represents client and server on different hosts).

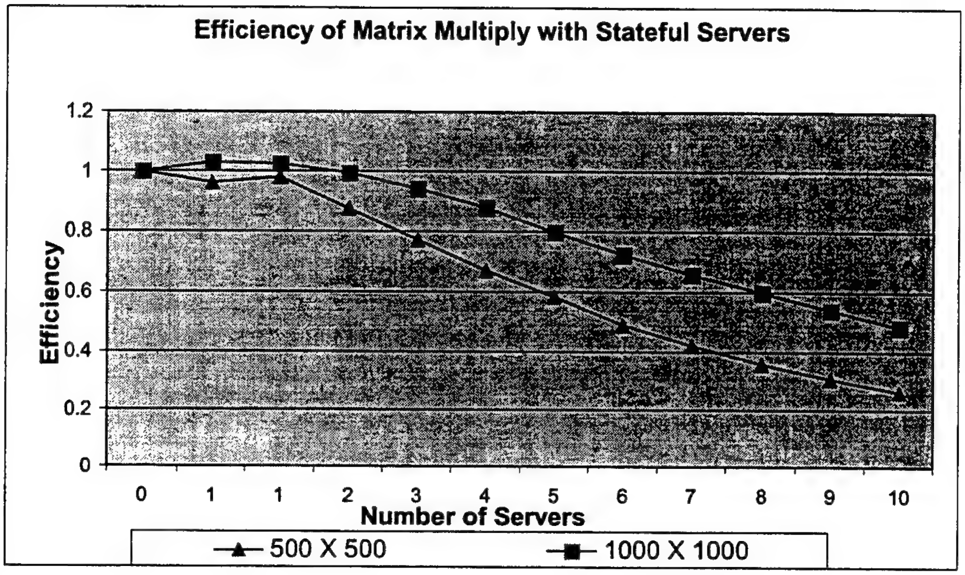


Figure 7: Stateful Server Efficiency
(Note: On the X axis, the first 1 represents client and server on same host, the second 1 represents client and server on different hosts).

C. Bugs and Lessons Learned

1) Achieving Speedup. Getting a sequential algorithm, even a simple one, to run faster on a distributed system is not as simple as we first thought. A lot of care in must be taken in how messages are passed to prevent creating a sequential algorithm with speedup less than 1.

2) Buffer Deadlock. We ran into a buffer deadlock problem in the stateless C++ implementation which took a while to find. The problem resulted from the sequential nature of the client program; the client would send all rows and matrices to all servers before ever receiving any messages. The servers replied immediately after calculating the result row, sending the reply back to the client. These replies piled up into the received buffer on the client side. At some point, the buffer became full and a server "hung" waiting to be able to send a message to the client. Meanwhile, the client is still writing messages across the socket. When it tries to send the message to the "hung" server, it is able to connect but not send the message, so it buffers this message. This continues until the client's buffer is full, and it hangs. The server is waiting for the client to read its messages, and the client is waiting for the server to read its messages, resulting in deadlock. We solved this by forking off the send portion of the client as a child process, so that the client can both send and receive at the same time.

3) Dynamically Allocating 2d Arrays in C++. You cannot dynamically allocate a 2d array directly in C++ (eg this declaration and definition is not valid:
int* myAry = new int[numRows][numCols]; where numRows and numCols are not known at compile time). To get around this, Matt implemented a 2D Array Class and Alpay allocated space using a "for" loop.

4) Corba Buffer Size. The Corba implementation had a limited buffer size; we could not pass more than a 200 X 200 matrix from client to servers (160,00 bytes). Using C++ sockets, we just had to statically allocate sufficient buffer space prior to run time.

5) TCP vs UDP. Since UDP does not guarantee delivery, or order of delivered packets, large messages (large arrays) could not reliably be reassembled, and TCP had to be used for the C++ sockets implementation.

6) Corba Server and Client on Same Machine. As discussed in paragraph XXX, when the Corba Server and Client ran on the same machine, it ran considerably slower than the C++ Socket client and server equivalent.

4. Source Code
   - A. C++ Sockets Program Listing
   - B. Corba Program Listing

5. Bibliography

1. Sean Baker, *CORBA Distributed Objects Using Orbix,* Addison-Wesley, 1997
2. John A. Zinky, David E. Bakken, and Richard D. Schantz,*Architectural Support for Quality of Service for CORBA Objects,* John Wiley & Sons, inc, 1997.
3. Robert Orfali, DanHarkey and Jeri Edwards,*Instant Corba,* John Wiley & Sons, inc, 1997.
4. Robert Orfali, DanHarkey and Jeri Edward *The Essential Distributed Objects Survival Guide*
   John Wiley & Sons, inc March 1996.
5. Robert Orfali and DanHarkey *Client/Server Programming with Java and CORBA* John Wiley & Sons, inc, 1997
6. IONA Technologies, *Orbix Programmer's Guide,* Iona Technologies, 1997
7. Hesham El-Rewini, Ted G. Lewis, *Distributed and Parallel Computing,* Manning, 1998.
8. John Shapely Gray,*Interprocess Communications in Unix, the Nooks and Crannies,,* Prentice Hall, 1997.

6. Appendix

A. C++ Sockets with Stateless Servers and Corba Results Table

Results of Distributed Matrix Multiply Using C/Unix Sockets, Stateless				
		Matrix Size		
Server Configuration	Iteration	10X10	100X100	200X200
no servers, only client	1	0.03422	0.213527	1.72539
	2	0.001418	0.21494	1.76686
	3	0.001334	0.210902	1.80662
	4	0.001368	0.216048	1.83943
	5	0.00136	0.214953	1.7988
	average	0.00794	0.214074	1.78742
1 server, same machine	1	0.02962	0.565334	3.19687
As client	2	0.060903	0.611573	3.20937
	3	0.026343	0.536085	3.227
	4	0.0328	0.53701	3.21336
	5	0.027317	0.535676	3.21543
	average	0.035397	0.5571356	3.212406
1 server, different machine from client	1	0.024698	3.69694	28.9662
	2	0.023073	3.70562	29.0327
	3	0.023013	3.68424	28.9536
	4	0.022988	3.69203	29.0587
	5	0.02301	3.70943	29.0125
	average	0.023356	3.697652	29.00474
5 servers, none on client machine	1	0.032801	3.71558	28.4544
	2	0.028046	3.724	28.4172
	3	0.02794	3.72808	28.4549
	4	0.039553	3.76833	28.4211
	5	0.030445	3.77034	28.4322
	average	0.031757	3.741266	28.43596
10 servers, none on client machine	1	0.029442	3.72934	28.558
	2	0.027547	3.71921	28.473
	3	0.041277	3.71126	28.549
	4	0.02741	3.7319	28.458
	5	0.028253	3.70922	28.545
	average	0.030786	3.720186	28.5166

Results of Distributed Matrix Multiply Using Corba				
		Matrix Size		
Server Configuration	**Iteration**	**10X10**	**100X100**	**200X200**
no servers, only client	1	N/A	N/A	N/A
	2	N/A	N/A	N/A
	3	N/A	N/A	N/A
	4	N/A	N/A	N/A
	5	N/A	N/A	N/A
	average			
1 server, same machine as client	1	0.100563	3.30899	30.9037
	2	0.105458	3.31681	28.8873
	3	0.111166	3.29292	29.3455
	4	0.118575	3.2985	30.1138
	5	0.095567	3.22849	29.0806
	average	0.106266	3.289142	29.66618
1 server, different machine from client	1	0.142726	4.52266	36.616
	2	0.10984	4.5406	36.0907
	3	0.119963	4.5302	36.564
	4	0.10807	4.53395	36.5332
	5	0.117222	4.49347	36.649
	average	0.119564	4.524176	36.49058
5 servers, none on client machine	1	0.162902	4.58567	36.396
	2	0.14028	4.57764	36.371
	3	0.156322	4.58606	36.2936
	4	0.168699	4.5618	36.4416
	5	0.171204	4.56622	36.32
	average	0.159881	4.575478	36.36444
10 servers, none on client machine	1	0.293782	5.18686	36.8186
	2	0.299528	5.26731	36.679
	3	0.24345	5.21294	36.6034
	4	0.271893	5.30495	36.5714
	5	0.251008	5.28372	36.6766
	average	0.271932	5.251156	36.6698

B. C++ Sockets with Stateful Servers Results Table

Results of Distributed Matrix Multiply Using C/Unix Sockets, Stateful					
		Matrix Size			
Server Configuration	Iteration	10X10	100X100	500 X 500	1000 X 1000
no servers, only client	1	0.03422	0.213527	33.287	338.103
	2	0.001418	0.21494	33.2161	340.052
	3	0.001334	0.210902	33.2668	337.83
	4	0.001368	0.216048	33.4291	338.653
	5	0.00136	0.214953	33.5073	337.354
	average	0.00794	0.214074	33.34126	338.3984
1 server, same machine as client	1	0.04	0.490709	34.7345	323.934
	2	0.023908	0.481969	34.7659	324.178
	3	0.024263	0.4661969	34.821	
	4	0.023914	0.468853	34.6721	
	5	0.023867	0.463636	34.6987	
	average	0.02719	0.4742728	34.73844	324.056
1 server, different machine from client	1	0.023526	0.415318	35.5373	326.585
	2	0.022547	0.421148	35.466	328.432
	3	0.020974	0.412318	35.389	
	4	0.021013	0.411463	35.4647	
	5	0.021036	0.414522	35.3565	
	average	0.021819	0.4149538	35.4427	327.5085
2 servers, none on client machine	1	0.029879	0.318277	18.9809	167.951
	2	0.026932	0.309274	19.0294	167.979
	3	0.02794	0.31011	19.1709	
	4	0.0281	0.31355	18.9653	
	5	0.02854	0.316789	19.0358	
	average	0.028278	0.3136	19.03646	167.965
3 servers, none on client machine	1	0.029879	0.41226	14.287	118.607
	2	0.029014	0.344178	14.2014	118.563
	3	0.027264	0.346746	14.4001	
	4	0.028554	0.34821	14.4802	
	5	0.029012	0.347826	14.4111	
	average	0.028745	0.359844	14.35596	118.585
4 servers, none on client machine	1	0.026567	0.406102	12.4153	95.012
	2	0.030197	0.486873	12.4224	95.1485
	3	0.037655	0.450747	12.4341	
	4	0.027786	0.405199	12.3933	
	5	0.02765	0.421101	12.401	
	average	0.029971	0.4340044	12.41322	95.08025

Results of Distributed Matrix Multiply Using C/Unix Sockets, Stateful

		Matrix Size			
Server Configuration	**Iteration**	**10X10**	**100X100**	**500 X 500**	**1000 X 1000**
5 servers, none on client machine	1	0.025989	0.442548	11.433	83.8903
	2	0.028055	0.464511	11.3503	83.8453
	3	0.026616	0.449753	11.4211	
	4	0.025807	0.460504	11.3711	
	5	0.0264	0.491189	11.436	
	average	0.026573	0.461701	11.4023	83.8678
6 servers, none on client machine	1	0.026368	0.490709	11.3172	76.807
	2	0.029012	0.481969	11.3876	76.742
	3	0.032566	0.4661969	11.1808	
	4	0.028125	0.468853	11.4787	
	5	0.029952	0.463636	11.333	
	average	0.029205	0.4742728	11.33946	76.7745
7 server, different machine from client	1	0.029589	0.606383	11.53	72.5306
	2	0.037985	0.593823	11.2737	72.3187
	3	0.028464	0.59655	11.3548	
	4	0.029125	0.605671	11.291	
	5	0.031254	0.599932	11.3025	
	average	0.031283	0.6004718	11.3504	72.42465
8 servers, none on client machine	1	0.034046	0.644748	11.5879	69.8434
	2	0.02762	0.635974	11.5782	69.8044
	3	0.033888	0.648489	11.59012	
	4	0.029773	0.599443	11.5855	
	5	0.03102	0.60215	11.5955	
	average	0.031269	0.6261608	11.58744	69.8239
9 servers, none on client machine	1	0.0473	0.633137	12.031	68.5766
	2	0.03555	0.626462	12.1045	68.8724
	3	0.0245	0.62955	12.059	
	4	0.02789	0.631859	12.078	
	5	0.02985	0.5981	12.068	
	average	0.033018	0.6238216	12.0681	68.7245
10 servers, none on client machine	1	0.050847	0.733338	12.5087	70.1243
	2	0.046994	0.702655	12.4678	68.4136
	3	0.042469	0.720646	12.5932	71.4136
	4	0.037012	0.719209	13.258	
	5	0.032846	0.693501	12.6363	
	average	0.042034	0.7138698	12.6928	69.26895

C. Speedup and Efficiency of Stateful Servers

Speedup, Stateful Servers				
	Matrix Size			
#Servers	10X10	100X100	500 X 500	1000 X 1000
0	1	1	1	1
1	0.292015	0.4513732	0.95978	1.04425902
1	0.45203	0.6219946	0.979477	1.03325074
2	0.280782	0.6826339	1.751442	2.01469592
3	0.276226	0.5949078	2.322468	2.85363579
4	0.264923	0.4932531	2.685948	3.55908193
5	0.298795	0.4636637	2.924082	4.03490255
6	0.271875	0.4513732	2.940286	4.40769266
7	0.253809	0.3565097	2.937452	4.67242023
8	0.253922	0.3418834	2.877361	4.84645515
9	0.240475	0.3431654	2.76276	4.9239849
10	0.188897	0.2998782	2.626785	4.88528266

Efficiency, Stateful Servers				
	Matrix Size			
#Servers	10X10	100X100	500 X 500	1000 X 1000
0	1	1	1	1
1	0.292015	0.4513732	0.95978	1.04425902
1	0.45203	0.6219946	0.979477	1.03325074
2	0.140391	0.341317	0.875721	1.00734796
3	0.092075	0.1983026	0.774156	0.95121193
4	0.066231	0.1233133	0.671487	0.88977048
5	0.059759	0.0927327	0.584816	0.80698051
6	0.045312	0.0752289	0.490048	0.73461544
7	0.036258	0.05093	0.419636	0.6674886
8	0.03174	0.0427354	0.35967	0.60580689
9	0.026719	0.0381295	0.306973	0.54710943
10	0.01889	0.0299878	0.262679	0.48852827

C. Machines Used in the Experiments

The machines used were Sun Ultra 10's, 300Mhz, in Spanagel #506.

Hosts used in experiment	
Host Name	**Role**
indus	Nameserver and client
lynx	server #1
mars	server #2
mensa	server #3
crater	server #4
ariel	server #5
apus	server #6
janus	server #7
gemini	server #8
grus	server #9
libra	server #10

E. Big "O" Analysis of Statefulvs Stateless Server Messages.

Assume Square, N X N matrices

Let N = # of rows and columns in both A and B matrices
S = # of servers to distribute to
C = constant value of index, and dimensions in work request (size of 3 longs)

Stateless Server Messages.

N messages are sent.

Each message contains an N-sized row, N^2 –sized matrix, and C-sized constants.

So, message complexity is $O(N)*O(N^2 + N + C)$, drop the constant term and multiplying yields $O(N^3)$.

Stateful Server Messages.

N messages sent.

The first message each server receives contains an N-sized row, N^2 –sized matrix, and C-sized constants. S of these messages are sent.

The subsequent messages each server receives contain an N-sized row, and C-sized constants. N – S of these messages are sent.

So, message complexity is $O(S)* O(N^2 + N + C) + O(N - S)*O(N + C)$. Recognize that S and C are constants and drop out of the equation yields $O(N^2 + N) + O(N^2)$, which simplifies to $O(N^2)$.

Passive, Domain-Independent, End-to-End Message Passing Performance Monitoring to Support Adaptive Applications in MSHN[ϒ]

Matt Schnaidt
Debra Hensgen
John Falby
Taylor Kidd
David St. John

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5118

Abstract

This paper focuses on the problem of monitoring the end-to-end performance of message passing to support adaptive applications to be executed using the MSHN system (Management System for Heterogeneous Networks). Eight commercial and research tools and application components that attempt to measure perceived end-to-end message passing performance were identified. Two were dismissed; one because of recently published findings and the other because it is typically used in too many inconsistent configurations. The remaining six are carefully described in the paper. We were able to characterize each as either passive or active, determine whether they require domain-specific knowledge of an application, identify sources of inaccuracies, and enumerate their limitations. Based upon this survey, and previous analytical experiments, we conclude that the optimal monitoring mechanism: (1) should be passive; (2) should not require domain-specific knowledge of an application; (3) should minimize sources of error; and (4) should have few limitations. No single tool or application component surveyed has all of these characteristics. Based upon the surveyed work and other recent research in distributed systems, we have synthesized a new tool whose mechanisms have all of the desired characteristics. This paper describes our mechanism, and how we implemented it, in detail.

1. Introduction

Any system managing a set of distributed heterogeneous resources, whether a native distributed operating system or a resource management system, must maintain status information concerning those resources. Some of the status information is slowly changing, such as the speed of a particular CPU. Some of the information may change more quickly, such as the version of the operating system, the type of security services,

[ϒ] This research is supported by DARPA Order E-583 under the DARPA QUORUM program and also, in part, by the Institute for Joint Warfare Analysis

and the type of network card. Finally, some status information will change very quickly such as the current length of the CPU queue, the amount of available memory, and the current load on a network. Though possibly cumbersome, the first two types of information can be manually entered into the management system's database when new hardware or software is installed. The third type of information, however, changes so quickly that there must be a method for automatically collecting it. The Management System for Heterogeneous Networks (MSHN[1]), an experimental distributed resource management system (RMS) that we are building, requires an estimate of this quickly changing information. Because users can place loads on some of the resources within MSHN's venue without submitting requests to MSHN, MSHN requires a mechanism that can accurately estimate the *end-to-end* availability of each of the resources. This paper documents our efforts to locate, and eventually, synthesize a mechanism to provide this information for the network resources. In this section, we first briefly describe the MSHN project, its place within the larger QUORUM program, and the reason why MSHN requires end-to-end status information. We then describe exactly what status information we currently need as well as the constraints under which a mechanism to obtain this information must operate.

1.1 MSHN

In order to put the research described in this paper into perspective we provide a brief overview of MSHN and we summarize the QUORUM program, under which MSHN is a project. The goal of the QUORUM program is to develop a software architecture, consisting of translucent layers, that delivers good end-to-end quality of

[1] Pronounced "mission"

service (QoS) to a dynamically changing set of adaptive applications that are competing for resources within a distributed, heterogeneous computing infrastructure. QUORUM consists of many major research projects including ones that: (1) define languages and models for expressing user-level QoS; (2) design and construct tools that convert user-level QoS to resource requirements; (3) design and construct languages and databases for describing resource requirements and resource characteristics; (4) design and construct RMS's; (5) define mechanisms for achieving translucence; (6) design appropriate feedback mechanisms; (7) define benchmarks to be used as representatives of future adaptive applications; and (8) research new ideas in distributed operating systems and network protocols. MSHN is one of several RMSs being designed, implemented, and tested under QUORUM. MSHN focuses on four basic areas: (i) the granularity of resources and the protocols and policies used to allocate them required by RMS's to derive good schedules for adaptive applications; (ii) heterogeneous scheduling algorithms; (iii) estimating the available resources as well as the required resources from historical and recently collected resource usage data; and (iv) determining how to ensure that RMSs remain in a stable state.

Given a set of jobs, MSHN will determine where and when to run each job along with the appropriate version of the job to run. MSHN evolved from SmartNet, which was a heterogeneous framework for minimizing the time at which the last job of a set of computationally intensive jobs finishes on a suite of heterogeneous computing resources [KIDD96]. SmartNet treated the set of compute resources available as one virtual heterogeneous machine (VHM). SmartNet achieved superior performance by mapping applications to resources based upon knowledge of the VHM and job characteristics.

MSHN differs from SmartNet in several ways: (1) it strives to support Input/Output intensive and real-time jobs, in addition to compute-intensive jobs; (2) it accounts for the fact that a job may need many different resources, not just a CPU, to execute; and (3) it manages adaptive applications.

One of the important improvements of MSHN over traditional RMS's is that MSHN will support adaptive applications. Adaptive applications are those that can produce results using one of a variety of algorithms or in one of a variety of forms.

MSHN has a client-server architecture. It is composed of a Client Library, a Scheduling Advisor, a Resource Requirements Database, a Resource Status Server, and a MSHN Daemon.

The following paragraphs provide an abstract description of each of the components, and Figure 1 provides an overview of the entire architecture. Although these components are shown together, they may in fact reside on separate machines, and, in certain situations, be replicated. Usually, many different client applications will be running at any given time.

The Client Library

The client library is linked with both adaptive and non-adaptive applications. It provides a transparent interface to all of the MSHN services [KRES97]. The client library performs at least the following functions: (1) it intercepts system calls to record resource requirements; (2) it forwards requests to start another process, when appropriate, to the Scheduling Advisor; and (3) it intercepts and performs the appropriate action on requests from the Scheduling Advisor to adapt. It forwards the recorded resource requirements to the Resource Requirements Database. The final implementation of MSHN will be able to forward the performance measurements and resource requirements through the MSHN daemon when that is more efficient.

The Scheduling Advisor

The Scheduling Advisor performs the highly complex task of scheduling multiple jobs, from multiple users, onto one (or several) computers from a pool of heterogeneous computing platforms. The sophisticated algorithms that the Scheduling Advisor will use to make decisions are beyond the scope of this paper. However, this research requires knowledge of the interfaces presented by the Scheduling Advisor.

The Scheduling Advisor will accept scheduling requests from the client libraries. The Scheduling Advisor will query both the Resource Status Server and the Resource Requirements Database. These queries must respond with near real-time information on the status (load) of the distributed resources, and the resource requirements of the application. Once the Scheduling Advisor receives this load information, it can calculate, if possible, a mixture of computing and network resources that will, with high probability, deliver the requested quality of service.

Additionally, in the event of a significant deviation from the initial resource status estimate, the Scheduling Advisor will receive notification from the Resource Status Server. For example, if a communications path is severed, or a machine fails, the Scheduling Advisor will be notified and can recalculate a new scheduling solution for the affected applications. The Scheduling Advisor may then signal the client library and advise it that the application should begin using a different algorithm, or perhaps recommend that it shift execution to a different set of resources. The granularity of resource model needed by the Scheduling Advisor is a topic of major research in the MSHN project.

The Resource Requirements Database

The Resource Requirements Database is a repository of information pertaining to the execution of user applications. This database contains statistics on the run time characteristics of jobs, such as CPU, memory, and disk usage. The Resource Requirements Database provides this information to the Scheduling Advisor upon request. While currently the MSHN client library is its only source of information, we envision that other tools, currently under development within the QUORUM project could also provide information for this database.

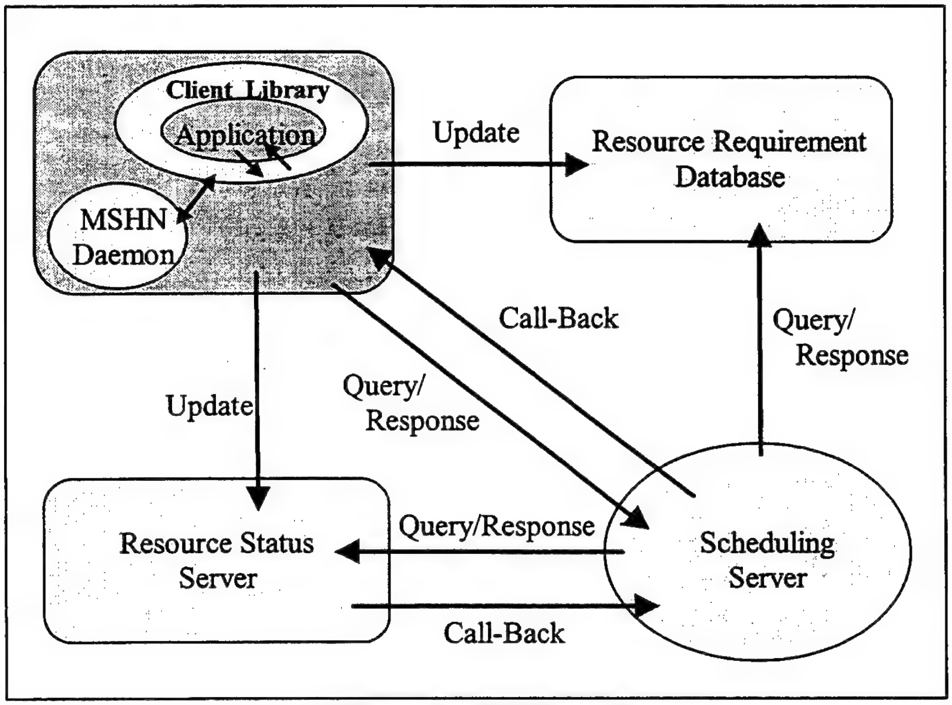


Figure 1: MSHN Architecture

The Resource Status Server

The purpose of the Resource Status Server is to maintain a repository of the three types of information about the resources available for MSHN to schedule: the relatively static, the moderately dynamic, and the highly dynamic information. The Scheduling Advisor will query the Resource Status Server to obtain an initial estimate of the currently available computing and networking resources. After making a scheduling decision, the Scheduling Advisor will notify the Resource Status Server of the additional loads that it expects the client application to place on the compute and networking resources. Much of this paper is dedicated to determining the best mechanisms for obtaining this most dynamically changing type of information for network resources.

The MSHN Daemon

The MSHN Daemon executes on all compute resources available for scheduling by the MSHN Scheduling Advisor. It is used to begin and control the execution of processes that are submitted to MSHN. It is also used to filter information from the client libraries which is destined for the Resource Requirements Database or Resource Status Server when several different MSHN jobs are executing locally.

1.2 Constraints

MSHN requires the gathering of resource usage information for applications that run within the MSHN system as well as status information for resources within the scope of the MSHN Scheduling Advisor. The MSHN Scheduling Advisor uses this information to make scheduling decisions. The methods used to gather this information are subject to three constraints: (1) the implementation must not require any changes to an operating system; (2) modifications to the application code must be minimized; and (3) the overhead imposed by the information gathering mechanism should not be excessive.

There are many reasons for the first requirement. Our ultimate goal is the widespread acceptance of MSHN-type systems. Many potential users are reluctant to use systems or tools that require that their operating system be modified. When routine operating system upgrades do occur, we do not want to have to redesign and redistribute our system to include the features or improvements of this new release. Addtionally, we do not want to risk compromising the security features of the operating system by changing the kernel. Finally, source code of all operating system releases may not be available.

The second and third requirements address acceptance and usability issues. If the application writer must modify his code, or if the use of our system incurs unacceptable overhead, the system will not be used.

1.3 Organization of Remainder of Paper

The next section of this paper surveys existing mechanisms that we considered as candidates for MSHN to use to estimate end-to-end network status. That section describes the six that we considered as initially viable candidates in some detail and summarizes their similarities and differences. In section 3, we describe the mechanism that we are currently using in MSHN, which synthesizes the best attributes that we found in our survey, as well as other recent theoretical and practical distributed system results. Finally, we summarize our findings – including giving an empirically obtained measurement of the worst-case overhead incurred and how we can reduce it – and outline the set of experiments that we are currently conducting to assess the accuracy with which we can predict quickly changing resource availability.

2. Existing Mechanisms for Estimating Network Availability

To support adaptive applications, the MSHN Scheduling Advisor requires end-to-end status information for the resources at its disposal. One critical resource is the network. The remainder of this paper addresses the issue of monitoring the current network availability. We first review several tools, protocols, and application components that estimate network availability and then summarize the desirable characteristics of these systems.

To support MSHN's end-to-end network monitoring, we initially considered eight application components, protocols, and tools, from both the commercial and research sectors: `Ping`, `ftp` (the File Transfer Protocol application), Netscape Communicator, Network Weather Service (NWS), Netperf, BBN's Communications Server (Commserver), Resource Reservation Protocol (RSVP) and Simple Network Management Protocol (SNMP). We quickly rejected two as inappropriate for MSHN: SNMP and RSVP. We could not directly use the Simple Network Management Protocol (SNMP) in MSHN because it provides link-based information and does not estimate end-to-end throughput and latency between machines on remote, nonadjacent networks [PERK97]. Any tool built on top of SNMP would necessarily have to change when routing algorithms changed. We also considered RSVP, but have rejected it largely due to recent results indicating that the bandwidth it allocates can be substantially different from that requested[LEEC98]. We now discuss each of the six remaining in detail and then compare and contrast advantages and disadvantages of each type.

2.1 Ping

The ping program is ubiquitous. Its primary use is for troubleshooting networks. In its default configuration, if a network connection exists between two machines, the execution of ping results in a short message making a round trip from the local to the remote host and back. When ping completes, it prints the number of bytes sent and the round trip time. Thus, ping provides a single packet measurement of network throughput.

Ping packages its local timestamp in an Internet Control Message Protocol (ICMP) packet and sends it to the targeted host. The targeted host receives this datagram, its IP layer recognizes the type (ECHO_REQUEST) and immediately repackages the data contained in the packet, sending it back to the pinger. The pinger receives the reply datagram and subtracts its machine's current time from the timestamp that the pinger placed in the original datagram, thus determining round trip time. [BERK91]

2.2 The File Transfer Protocol Application

The File Transfer Protocol application (ftp) is used to transfer files between machines connected by a network. We studied ftp because it estimates end-to-end (meaning application to application) throughput after completing a file transfer. The ftp application is a client-server application; the ftp server runs as a background process listening on a fixed port for client connection requests. The user starts the ftp client in order to issue requests to the ftp server on the remote machine. Because of the tight coupling between the ftp server and client, it is possible for ftp to estimate the throughput associated with transferring a file. We now summarize the actions that occur in transferring a file, F, from the ftp server on computer A to the ftp client on computer B.

The ftp server on computer A listens on port Y. The ftp client on computer B connects to the server on computer A at port Y. The server on computer A accepts the connection and generates a child process that will handle all future interactions with the ftp client via the connection. We will call this connection the control connection. The ftp client on computer B sends a request across the control connection to the ftp server on computer A to send file F. Included in the request to A is the port number, X, that the client on computer B will listen to. The ftp client on B then listens on port X. The server sends the size, S, of file F to the ftp client over the control connection. The client receives S. The server connects to the client on computer B at port X. The client accepts the connection and records computer B's time as $T_{startRead}$. We will call this connection the data connection. Once the server sees that the client accepted the data connection, it sends the file, F, across that connection and the client reads until it receives the entire file. When the entire file has been received, the client records computer B's time as $T_{endRead}$. The client then uses S and the time elapsed between $T_{startRead}$ and $T_{endRead}$ to estimate throughput.

2.3 Netscape Communicator

Netscape recently made the source code for their web browser freely available [NETS98]. By examining this source code, we learned that they use an approach similar to ftp's to calculate throughput. Netscape Navigator displays a "thermometer" at the bottom of the browser, showing the user the current download speed, the amount of the file already downloaded, and the estimated time to complete the download. When downloading large files, the system call `read()` must typically be invoked more than once by the browser. Following each invocation of `read()`, Netscape updates the thermometer's data fields. The throughput estimate displayed on the thermometer is cumulative in that each calculation is based upon the total number of bytes downloaded, the total size of the file to download, and the total time since the first `read()` was called for the current file.

2.4 Network Weather Service

The Network Weather Service (NWS) is a tool for predicting computer and network performance for use by metacomputing applications [WOLS97] [SPRI97]. NWS makes periodic estimates of availability of resources for which it is responsible. One of the estimates made by NWS is network availability, specifically, the latency and throughput observed between two computers.

In order to measure the latency between two computers, A and B, a NWS process on computer A sends a small message to a corresponding process on computer B. The process on B immediately replies to the process on A, with the process on A recording the round trip time. NWS approximates latency as half of this round trip time.

To estimate throughput, the NWS process on host A sends a large message to the corresponding process on host B, and the process on B sends a small acknowledgement message back to the process on A. The NWS process on computer A estimates the transmission time of the large message as the round trip time, less the latency estimate described above. Throughput is then estimated as the message size divided by estimated transmission time. NWS keeps a record of previous estimates of throughput and latency, which it uses to predict future resource availability using statistical modeling techniques.

The developers use a token passing scheme to avoid overloading the network. They note that token passing does not scale well with large distributed systems. Therefore, the accuracy of the throughput and latency estimates degrade as the size of the network increases. Additionally, token passing can introduce security problems [STAL98]. Finally, to capture fluctuations in network QoS, the developers note that administrators must increase both message size and monitoring frequency.

2.5 Netperf

Netperf is a benchmark that can be used to measure different aspects of network availability with a primary focus on actively measuring the throughput of bulk data transfers and the request/response round trip time [HEWL96]. Netperf contains a rich benchmarking suite with many options for simulating specific scenarios (e.g., http protocols). Netperf's bulk data transfer test can be used to estimate the throughput between a local and remote host communicating over a network. It works by sending data for a period of time (the default is 10 seconds), and then measuring the total amount of data sent and received after the time period has elapsed. Netperf's request/response time test is very similar to that used by NWS to estimate latency: a short message is sent from a local to a remote host; the remote host replies immediately; and the local host measures round trip time and approximates latency as one half of this round trip time.

2.6 BBN's Commserver

The Joint Task Force Advanced Technology Demonstration (JTF ATD) strives to predict trends in the advances of future hardware and software. It also provides a reference architecture into which such advances can be easily integrated. At the base of the JTF architecture is the Commserver whose ultimate purpose is to permit applications to be network aware. The job of the Commserver is to estimate the available bandwidth between JTF users. The Commserver uses that bandwidth, along with the priorities of various users (expressed as currency), and a list of the applications requiring execution to allocate resources. [HAYE94]

Early experimentation with the JTF ATD Commserver [KRES98] revealed several problems. First, the Commserver places a load on the network in order to estimate end-to-end latency and bandwidth available. Second, it uses the throughput and latency estimates directly, without reference to previous measurements, to estimate network load. Due to the rapidly changing nature of network traffic, this technique can return inaccurate or misleading results. Finally, the estimates returned are inaccurate unless the network is sampled frequently which further increases the network load.

2.7 Characteristics of These Systems

We divide the mechanisms described above into two categories: **passive** and **active**. Active mechanisms place additional loads on the resources that they are monitoring; passive ones do not. Applying this definition, we see that `ping`, NWS, Netperf and BBN's Commserver all use active mechanisms, while `ftp` and Netscape use passive mechanisms. Unfortunately, it is when the network is most busy that we need the most accurate estimates. This is when there is no extra bandwidth available to give to these active mechanisms. Passive mechanisms, on the other hand, do not add to the load carried by the already scarce resource. For this reason, in MSHN we prefer passive monitoring.

Another way of categorizing the previously discussed tools and application components for measuring network performance attributes is to consider how closely tied they are to applications. We note that the programmers of both ftp and Netscape used domain-specific knowledge to obtain estimates of network throughput. MSHN prefers that application writers not be required to also worry about measuring resource availability; availability should be measured by the system.

Finally, there are sources of error and limitations associated with the previously described mechanisms. When estimating throughput, all mechanisms start timers with a handshake. The best ones then subtract off some multiple of an estimate of latency, which they assume to be the amount of time required for the handshake, but which may actually be substantially different due to operating system CPU scheduling policies. Further, none of the passive monitoring techniques estimate latency, only throughput. The MSHN system requires, at a minimum, the knowledge of both of these.

Based on these observations, we sought a passive mechanism that would accurately estimate both bandwidth and latency without requiring the application programmer to implement monitoring code. In the remainder of this paper we describe such a mechanism.

3. A Passive Approach for Monitoring Network Performance

Before describing our domain-independent mechanism for passively obtaining accurate network performance estimates, we enumerate some of the challenges we faced in evolving such a mechanism.

3.1 Challenges

In order to avoid modifying either the operating system or the system libraries, we chose to apply a technique developed by Condor [LIVN95]. That is, we chose to implement an additional library that intercepts `read()` and `write()` calls and then link applications' object code with that library.[2] This additional library will then be able to pre-process parameters of `read()` and `write()` and post-process the results. We will define this interception of system calls as **wrapping** system calls.

After identifying the mechanism required to implement "domain-independence" and "passive monitoring" we turned our attention to the problem of accurately estimating bandwidth and latency. In the remainder of this section we refer to a process that issues a `write()` call to write across the network as the "writer" and the process that issues the corresponding `read()` as the "reader." In this paper, we also assume that the reader and writer are using TCP.

In order to estimate latency, we must know when the writer writes the message, and when the reader's computer receives it. We face several problems in trying to accurately obtain these times. First, the reader does not know when the writer wrote the message to the network. Second, if we modify `write()` so that the writer appends its local time to the beginning of the message, we still must compensate for the clock offset between the reader and writer computers. Since these computers do not have synchronized clocks, we cannot directly compare the writer's send time to the reader's receive time. Finally, if the writer writes the message long before the reader is ready to read the message, the message will be buffered on the reader's machine. This makes it difficult to estimate the time of reception. We will discuss these problems in some depth after summarizing the corresponding problems associated with estimating throughput.

In estimating throughput, we face similar challenges. Because we do not have control over the operating system, we have difficulty estimating transmission time. This

[2] If object code is not available, techniques developed in Paradyn [PARA97] [LARU95] could be used to link this library with the executable.

affects our ability to estimate throughput. From an application's perspective, once it calls `read()`, it blocks and remains blocked until the operating system returns with data in the buffer. We could measure the total blocked time after an application makes a `read()` system call and assume that this elapsed time is an estimate of total transmission time. However, unless the `write()` that corresponds to the `read()` occurred at the same time, this assumption would most likely result in incorrect throughput estimates because of the composition of the blocked time.

We refer to two significant problems associated with estimating transmission time as the "early reader-late writer" problem and the "late reader-early writer" problem. The remainder of this section will further explain these problems.

In the "early reader-late writer" scenario, the reader calls `read()` and blocks waiting for the writer to execute `write()`. Some time after the writer finally writes, the reader receives the message and unblocks. In this case, the total blocked time is composed of both the time spent transmitting as well as the time spent waiting for the late writer. Because we cannot know how much of the blocked time was due to waiting for the late writer, we cannot assume that blocked time is equivalent to the transmission time. If we were to make such an assumption, we would underestimate throughput.

In the "late reader-early writer" scenario, the writer writes to the network on an established connection, but the reader has not yet called `read()`. The operating system on the reader's host may buffer some or all of the data received from the writer. When the reader finally calls `read()`, it reads the buffered portion of the message directly from local memory. In this case, the total blocked time is composed of time spent reading from local memory, as well as the time required to read the unbuffered portion of the message from the network. In most systems, retrieving data from memory is significantly faster than network transmission time, so using this total time would result in an overestimate of throughput.

In summary, unless the `read()` and `write()` calls happen at exactly the same time, we cannot simply use blocked time to estimate throughput. Additionally, we face problems related to clock offset in estimating latency. Our approach to measuring network QoS will recognize and take advantage of the “early reader-late writer” scenario to aid in obtaining accurate latency and throughput estimates.

3.2 An Additional Observation That Helps

Many writes to the network by applications are large (e.g., files and graphics). As we saw in Netscape, a `read()` from the network returns immediately upon receiving data in the buffer. Even though an application writer specifies the amount to be read in the `read()` system call, the call will return with the amount of data actually read immediately upon receiving any data. To ensure that all desired data is read, the application writer must implement the application so that it repeatedly calls `read()` until it has read the entire message. We will use this observation in conjunction with the “early reader-late writer” scenario to help estimate throughput.

3.3 Our Passive Monitoring Approach

In this section we describe the passive network monitoring approach that we developed for MSHN. We will give an overview of our approach and then address the following four areas: clock offset compensation, the cooperating writer, the cooperating reader, and special considerations.

Overview of the MSHN Passive Network Monitoring Approach

Our approach exploits the "early reader-late writer" scenario. We have wrapped the `read()` and `write()` library calls to recognize TCP connections. In our approach, the `read()` system call recognizes the "early reader-late writer" scenario, allowing the estimation of end-to-end latency, and when appropriate, throughput.

In measuring end-to-end latency, we must address the three problems raised above. The first is that we do not know when the writer sent its message. We solve this by wrapping the `write()` system call to append, to the front of the message, a timestamp from the writer's clock. The second problem results from the fact that the reader's clock and the writer's clock are not synchronized, but the reader and writer need to have reference to a common timeline. We will address this problem in the next subsection. The final problem deals with the "late reader-early writer" scenario. We avoid this problem by detecting this situation and only calculating latency when we are sure that we are in the "early reader-late writer" scenario.

To estimate throughput, our mechanism must first estimate transmission time. In the "early reader-late writer" scenario, blocked time is composed of the end-to-end transmission time and the time spent waiting for the writer. We will exploit our observation that many network writes are large. As mentioned above, when a large message is read from the network, `read()` must be called repeatedly until the entire message is received. The two necessary components for calculating throughput are the number of bytes transmitted and transmission time for those bytes. Our algorithm computes the difference between the times of the first read and the last read. This difference is transmission time. Our technique "throws away" that first period of blocked time consisting of time due both to end-to-end data transmission as well as time spent waiting for the writer. It is safe to assume that the remaining time that it takes to read the message is due primarily to transmitting the remainder of the message. The number of bytes received will be the message size less the size returned by the first `read()`. This allows us to estimate throughput between the reader and writer.

We use the term “cooperative” to refer to the reading and writing applications that are linked with our library. We discuss cooperative readers and writers in more detail shortly.

Compensating for Clock Offsets

Our passive network monitoring requires that the communicating hosts have access to a common time reference. Since we cannot assume that the member hosts of MSHN have perfectly synchronized clocks, we use a derivative of the Network Time Protocol (NTP) to estimate clock offsets between machines [COUL96]. Our approach is almost identical to the protocol as described in the reference, with the exception that we eliminate the need for one of the timestamps. As the reference does, we call the estimated clock offset o_i and the estimated error, $d_{i.}$ In this protocol, shorter round trip times result in smaller errors. We exploit this observation by making multiple calls to the remote timeserver, and then keeping the offset that results from the shortest round trip time.

In MSHN, we would expect estimates of o_i and d_i to be available from the Resource Status Server (RSS), but prior to adding this functionality to the RSS, we implemented and tested our passive network performance monitoring algorithm by wrapping the `accept()` and `connect()` system calls to trigger these estimates. This extra code adds considerable overhead to these system calls. In the final implementation of MSHN the RSS would periodically, at times of low system usage, poll MSHN members and record clock offset and drift. To minimize added network traffic for delivery, the distribution of o_i and d_i can be included with security certificates [WRIG98].

The Cooperating Writer

We incorporated, into the MSHN library, a wrapper for the `write()` system call that detects when `write()` has been called to write to a TCP connection. The wrapper appends the following information to the front of such messages: the writer machine's current time, $T_{remote'}$, and the size of the message.

The Cooperating Reader

Like the write() system call's wrapper, the read() system call's wrapper also detects when it is reading from a TCP connection. We now enumerate the steps that the wrapper takes if it detects that this is the first time that the `read()` is being called for the particular data set:

1. It records a local clock time stamp, $T_{blocked}$, prior to (possible) blocking.
2. When the `read()` continues (unblocks), the wrapped system call records the time, $T_{startRead}$.
3. $T_{remote'}$ and total message size are stripped from the first part of the message received.
4. $T_{remote'}$ is adjusted for clock offset between the reader and writer machines and this adjusted time is recorded as T_{remote}.
5. `read()` now tests to see whether the "early reader-late writer" situation has occurred. If $T_{blocked}$ occurred earlier than T_{remote}, then the reader was early. That is, the reader was blocked for a while waiting on the writer to write. Only in this case can latency be approximated.

   $$\text{Latency} = T_{startRead} - T_{remote}.$$

6. The received message data is passed to the application, with the return value of the `read()` system call decremented to account for the size of the timestamp and total message size fields.

We now describe the actions taken when the read wrapper is invoked after the initial `read()`.

1. The size of the message remaining is decremented by the amount of data in the buffer.

2. If the size of the message remaining is zero, the end of the message has been found. In this case, throughput can be calculated.

3. The `read()` wrapper calculates throughput using:

Throughput = (total message size – size of first part of message)/($T_{endRead}$ -$T_{startRead}$)

We note that throughput is only calculated when more than one read() is invoked to obtain the data that was sent. We also note that the throughput and latency estimates are estimates of end-to-end throughput, which are, in fact, what MSHN is interested in.

Special Consideration

Latency and throughput can only be calculated for a subset of the total network traffic, that is when the "early reader-late writer" scenario is true. We can modify the algorithm to increase the opportunities to estimate throughput by "loosening" the "early reader-late writer" requirement if we have a good estimate of the absolute minimum latency, Latency_{min} between the communicating machines. Rather than requiring $T_{blocked}$ to occur before T_{remote} (that is, $T_{blocked} - T_{remote} < 0$), we could allow $T_{blocked}$ to be up to the minimum latency later than T_{remote} ($T_{blocked} - T_{remote} < \text{Latency}_{min}$). This seems insignificant for machines connected locally over high speed networks where latency is in the order of milliseconds or fractions of milliseconds. However, consider machines connected over extended network links, especially those using satellite communication. In this case, latency is in the order of 100's of milliseconds and this loosened requirement could prove significant. In this latter case, the opportunities to estimate throughput could increase dramatically with this modification.

4. Summary

In our review of existing tools and application components, we classified the network monitoring techniques as either passive or active. Passive monitoring has the desirable attribute of minimal added overhead while active monitoring gives the ability to measure latency and is not tied to any particular application. We then presented an approach that makes use of the low overhead of passive monitoring, estimates end-to-end latency and throughput, and is independent of any particular application. Preliminary experiments show that our technique adds 6% to the required execution time of the `read()` system call, confirming the low overhead of passive monitoring. We anticipate that using the tools developed by Oregon Graduate Institute's Synthetix team could further reduce this overhead [PUCA96].

We plan further work to quantitatively assess the accuracy of predictions based upon our mechanism.

5. Acknowledgements

We gratefully acknowledge Professor Cynthia Irvine's suggestion that we use the number of bytes to be sent rather than start and end flags. Although this solution is not guaranteed to work with every implementation of write, when it does work, it certainly reduces the overhead that we encounter when using start and end flags and bit stuffing.

6. References

[COUL96] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems, Concepts and Designs, 2d Edition*, Addison-Wesley, NY, 1996.

[BERK91] Berkeley Unix Distribution, *Unix Man Pages*, March 1991.

[HAYE94] G. Hayes-Roth, and L. Erman, *The Joint Task Force Architecture Specification (JTFAS)*, Teknowledge Federal Systems, Palo Alto, CA, 1994.

[HEWL96] Information Networks Division, Hewlett-Packard Company, *Netperf: A Network Performance Benchmark, Revision 2.1*, February 1996.

[KIDD96] T. Kidd, D. Hensgen, R. Freund, L. Moore, "SmartNet: A Scheduling Framework for Heterogeneous Computing", *ISPAN*, 1996.

[KRES97] J. Kresho, *Quality Network Load Information Improves Performance of Adaptive Applications*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1997.

[KRES98] J. Kresho, D. Hensgen, T. Kidd, and G. Xie, *Determining the Accuracy Required in Resource Load Prediction to Successfully Support Application Agility*, EURO-PDS98, 1998.

[LARU95] J. Larus and E. Schnarr, *EEL:Machine-Independent Executable Editing,* SIGPLAN PLDI 1995.

[LEEC98] C. Lee, J. Stepanek, B. Michel, I. Foster, C. Kesselman, R. Lindell, S. Hwang, J. Bannister, and A. Roy, *Qualis: the Quality of Service Component for the Globus Metacomputing System*, IWQoS98, Napa, CA, 1998.

[LIVN95] M. Livny, M. Litzkow, T. Tannenbaum, and J. Basney, *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*, Dr Dobbs Journal, February 1995.

[NETS98] Netscape Communication's Corporation, *Netscape Communicator Source Code,* March 1998.

[PARA97] Paradyn Project, *Paradyn Parallel Performance Tools User's Guide Release 2.0*, University of Wisconsin-Madison, 1997.

[PERK97] D. Perkins, and E. McGinnis, *Understanding SNMP MIBs*, Prentice-Hall, NJ, 1997.

[PUCA96] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole, and K. Zhang, *Optimistic Incremental Specialization: Streamlining a Commercial Operating System*, Oregon Graduate Institute, 1996.

[SILB98] A. Silberschatz, and P. Bae Galvin, *Operating Systems Concepts, 5th Edition*, Addison-Wesley, Menlo Park, CA, 1998.

[SPRI97] N. Spring, *Network Weather Service for Mentat 3.0 User's Guide*, October 1997.

[STAL98] W. Stallings, *Cryptography and Network Security, Principles and Practice, 2nd Edition*, Prentice Hall, Upper Saddle River, NJ, 1998.

[WOLS97] R. Wolski, N. Spring, and C. Peterson, *Implementing a Performance Fore-casting system for Metacomputing: The Network Weather Service*, SC97 Technical Paper, 1997.

[WRIG98] R. Wright, D. Shifflett, and C. Irvine, *Security for a Virtual Heterogeneous Machine*, to appear in Proceedings of the 12th CSA Conference, Scottsdale, AZ, 1998.

Techniques for Mapping Tasks to Machines in Heterogeneous Computing Systems

Howard Jay Siegel and Shoukat Ali

Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285 USA
Email: {hj, alis}@ecn.purdue.edu

June 1999

Invited Keynote Paper for the Journal of Systems Architecture
Special Issue on Heterogeneous Distributed and Parallel Architectures: Hardware, Software and Design Tools

Keywords: distributed computing, heterogeneous computing, metacomputing, resource management system, scheduling.

Abstract

Heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize their combined performance and/or cost-effectiveness. HC systems are becoming a plausible technique for efficiently solving computationally intensive problems. The applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources. In an HC system, tasks need to be matched to machines, and the execution of the tasks must be scheduled. The goal of this invited keynote paper is to: (1) introduce the reader to some of the different distributed and parallel types of HC environments; and (2) examine some research issues for HC systems consisting of a network of different machines. The latter purpose is pursued by considering: (1) the quantification of heterogeneity; (2) the characterization of techniques for mapping (matching and scheduling) tasks on such systems; (3) an example HC resource management system; and (4) static and dynamic heuristics for mapping tasks to machines in such HC systems.

This research was supported by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and under GSA subcontract number GS09K99BH0250. Some of the equipment used in this research was donated by Intel and Microsoft.

1. Introduction

In general, heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize their combined performance and/or cost-effectiveness [MaB99, Esh96]. HC systems are becoming a plausible technique for efficiently solving computationally intensive problems [FoK99]. The applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources. As machine architectures become more advanced to obtain higher peak performance, only a fraction of this performance may be achieved on many real tasks because a typical task may have various subtasks with different architectural requirements. When such a task is executed on a given machine, the machine may spend much of its time executing subtasks for which it is unsuited [Fre89].

One way to exploit an HC environment is to decompose a task into subtasks, where each subtask is computationally well suited to a single machine architecture, but different subtasks may have different computational needs (e.g., [WaS98]). These subtasks may share stored or generated data, creating the potential for inter-machine dependencies and data transfer overhead. Once the subtasks are obtained, each subtask is assigned to a machine (matching). Then the subtasks and any inter-machine data transfers are ordered (scheduling) so as to optimize some objective function. The overall problem of matching and scheduling is referred to as mapping. The objective function can be the overall completion time of the task or a more complex function of multiple requirements.

In some cases, a collection of independent tasks must be mapped, instead of a set of inter-dependent subtasks. Such an independent set of tasks is called a meta-task [FrK96]. An example of meta-task mapping is the mapping of an arbitrary set of independent tasks from different users waiting to execute on a heterogeneous suite of machines. Each task in the meta-task may have associated requirements, such as a deadline and a priority.

One broad objective of the HC community is to design a management system for HC resources (machines, networks, data repositories, etc.) [MaB99]. One important issue within this arena is the design of a mapping system that makes good decisions. Such a system, which may be called a scheduling advisor, has to be provided with an objective function that it tries to optimize. Current research involves formulating an optimization criterion that will be a function of a set of quality of service (QoS) attributes that are likely to be requested by the tasks expected in a given HC environment [KiH99]. This optimization criterion also will serve as a measure of the performance of the various scheduling approaches that might be available to the community, and also for that of the resource management approaches in general.

The scheduling advisor might have to choose between static and dynamic approaches to the mapping of tasks. Static approaches are likely to suffice if the the tasks to be mapped are known beforehand, and if the predictions about the HC resources are likely to be accurate. Dynamic approaches to mapping are likely to be more helpful if the HC system status can change randomly, and if the tasks that are supposed to be mapped cannot be determined beforehand. The general problem of developing an optimal matching of tasks to hosts is NP-hard [Fer89]. The goal of this invited keynote paper is to: (1) introduce the reader to some of the different distributed and parallel types of HC environments; and (2) examine some research issues for HC systems consisting of a network of different machines. The latter purpose is pursued by considering: (1)

the quantification of heterogeneity; (2) the characterization of techniques for mapping tasks on such systems; (3) an example HC resource management system; and (4) static and dynamic heuristics for mapping tasks to machines in such HC systems.

Section 2 briefly describes some broad classes of HC systems. In Section 3, "mixed-machine" HC systems are further classified and elaborated. Section 4 characterizes heuristics for mapping independent tasks onto a class of HC systems. An example system for managing resources in HC systems is discussed in Section 5. Sections 6 and 7 define and compare some dynamic and static mapping heuristics, respectively. Section 8 concludes the paper.

2. Parallel and Distributed Heterogeneous Computing Systems

There is a great variety of types of parallel and distributed heterogeneous computing systems. In this section, three broad classes are briefly described: mixed mode, multi-mode, and mixed-machine. The rest of the paper focuses on mixed-machine systems.

A mixed-mode HC system refers to a single parallel processing system, whose processors are capable of executing in either the synchronous SIMD or the asynchronous MIMD mode of parallelism, and can switch between the modes at the instruction level with negligible overhead [SiM96]. Thus, a mixed-mode machine is *temporally* heterogeneous, in that it can operate in different modes at different times. This permits different modes of parallelism to be used to execute various portions of a program. The goal of mixed-mode HC systems is to provide in a single machine the best attributes of both the SIMD and the MIMD models. PALM, TRAC, OPSILA, Triton, and EXECUBE are examples of mixed-mode HC systems that have been prototyped [SiM96].

There are various trade-offs between the SIMD and MIMD modes of parallelism, and mixed-mode machines can exploit these by matching each portion of a given program with the mode that results in the best overall program performance. Studies have shown that a mixed-mode machine may outperform a single-mode machine with the same number of processors for a given program (e.g., [FiC91]).

Multi-mode heterogeneous computing is similar to mixed-mode HC in the sense that multiple modes of computation are provided within one machine. However, it is different because all modes of computation can be used simultaneously. An example multi-mode architecture is the image understanding architecture (IUA) [WeL89]. In IUA, heterogeneity is incorporated by having multiple processing layers, where each layer provides a different form and mode of computation. Two levels of MIMD and one level of SIMD processors are included in this system.

Thus, mixed-mode and multi-mode systems represent one extreme of HC, where the heterogeneity resides in a single machine. For more about such systems, see [Esh96].

In mixed-machine HC systems, a heterogeneous suite of machines is interconnected by high-speed links to function as a metacomputer [KhP93] or a grid [FoK99]. (The grid refers to a large-scale pooling of resources to provide dependable and inexpensive access to high-end computational capabilities [FoK99].) A mixed-machine HC system coordinates the execution of various components of a task or a meta-task on different machines within the system to exploit the different architectural capabilities available, and achieve increased system performance [MaB99].

3. Degrees and Classes of Mixed-Machine Heterogeneity

In a mixed-machine HC system, each task can have a different execution time on each machine. A heuristic is employed to map tasks onto the machines in an HC system. The assumption that estimates of expected task execution times on each machine in the HC suite are known is commonly made when studying mapping heuristics for HC systems (e.g., [GhY93]). (Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [MaB99].) These estimates can be supplied before a task is submitted for execution, or at the time it is submitted. The mapper contains an ETC (expected time to compute) matrix that contains the expected execution times of a task on all machines, for all the tasks that are expected to arrive for service. In an ETC matrix, the elements along a row indicate the execution times of a given task on different machines, and those along a column give the execution times of different tasks on a given machine. The average variation along the rows is referred to as the machine heterogeneity; this is the degree to which the machine execution times vary for a given task, averaged over all the tasks [Arm97]. Similarly, the average variation along the columns is referred to as the task heterogeneity; this is the degree to which the task execution times vary for a given machine, averaged over all the machines in the system [Arm97].

Based on the above idea, four categories were proposed for the ETC matrix in [Arm97]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo). The ETC matrix can be further classified into two classes, consistent and inconsistent [Arm97], which are orthogonal to the previous classifications. For a consistent ETC matrix, if machine m_x has a lower execution time than machine m_y for task t_k, then the same is true for any task t_i. The consistent case represents the situation where there is a definite ordering among the compute power of the machines in the suite. In inconsistent ETC matrices, the relationship among the execution times for different tasks on different machines is random. This is the opposite extreme of the consistent case. The inconsistent case represents a mix of task computational requirements and machine capabilities such that no ordering as that in the consistent case is possible. A combination of these two cases, which may be more realistic in many environments, is the semi-consistent ETC matrix, which is an inconsistent matrix with a consistent submatrix. As an example, in a given semi-consistent ETC matrix, 50% of the tasks and 25% of the machines may define a consistent sub-matrix. Furthermore, it may be assumed that for a particular task the execution times that fall within the consistent sub-matrix will be smaller than those that fall out. These degrees and classes of mixed-machine heterogeneity can be used to characterize HC environments.

4. Characterizing Mapping Heuristics

The mapping of tasks and meta-tasks, and the scheduling of communications in HC environments, are active, growing areas of research. A taxonomy of mapping heuristics is useful to researchers as it allows meaningful comparison among different mapping heuristics used in different HC environments for different applications. The Purdue HC Taxonomy [BrS98] is a three part classification that attempts to classify mapping heuristics according to the features of the applications being mapped (i.e., application model), characteristics of the hardware that the mapper is targeting (i.e., HC hardware platform model), and mapping strategies (i.e., mapper model).

The Purdue HC Taxonomy classifies the application being mapped on the basis of latter's size, divisibility into (possibly dependent) subtasks, communication patterns, quality of service requirements, etc. The taxonomy distinguishes among hardware platforms on the basis of communication time estimates between different machines, possibility of concurrency in sends and receives, machine architecture and heterogeneity, interconnection network, etc. Similarly, mapping heuristics are classified on the basis of their ability to adapt to changes in HC system, support various application models, consider subtask data dependencies and communication times, fault tolerance, etc.

A researcher also can use the taxonomy to find mapping heuristics that use similar target platform and application models. The mapping heuristics found for similar models can then possibly be adapted or developed further to better solve the mapping problem under consideration.

5. MSHN: An Example Resource Management System

5.1. Overview

A resource management system (RMS) views the set of heterogeneous machines that it manages as a single virtual machine, and attempts to give the user a location-transparent view of the virtual machine [ReT85]. The RMS should be able to provide the users a higher level of overall performance than would be available from the users' local system alone.

The Management System for Heterogeneous Networks (MSHN–pronounced "mission")

[HeK99] is an RMS for use in HC environments. MSHN is a collaborative research effort that includes the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. It builds on SmartNet, an operational scheduling framework and system for managing resources in an HC environment developed at NRaD [FrG98].

The technical objective of the MSHN project is to design, prototype, and refine a distributed RMS that leverages the heterogeneity of resources and tasks to deliver the requested QoS. To this end, MSHN is investigating: (1) the accurate, task-transparent determination of the end-to-end status of resources; (2) the identification of different optimization criteria and how non-determinism and the granularity of application and platform models (as outlined by the Purdue HC Taxonomy [BrS98]) affect the performance of various mapping heuristics that optimize those criteria; (3) the determination of how security should be incorporated within components as well as how to account for security as a QoS attribute; and (4) the identification of problems inherent in application and system characterization.

5.2. MSHN Architecture

Figure 1 shows the conceptual architecture of MSHN. As can be seen in the figure, every task running within MSHN makes use of the MSHN Client Library (CL) that intercepts the task's operating system calls. The Scheduling Advisor (SA) determines which set of resources a newly arrived task (or equivalently, a newly started process) should use. (Using the terminology from Section 1, the SA is a mapper.) The Resource Status Server (RSS) is a quickly changing repository that maintains information concerning the current availability of resources. Information is stored in the RSS as a result of updates from both the CL and the SA. The CL can update the RSS as to the currently perceived status of resources, which takes into account resource loads due to processes other than those managed by MSHN. The Resource Requirements Database (RRD) is responsible for maintaining information about the resources that are required to execute a particular task. The RRD's current source of information about a task is the data collected by the CL from the previous runs of the task. The RRD has the ability to maintain very fine grain experiential information collected by the CL, and it is hoped that, in future, it can also be populated with information from smart compilers and directives from task writers.

When the CL intercepts a request to execute a new task, it invokes a scheduling request for that task on the SA (assuming that the task requests to be scheduled through the SA). The SA queries both the RRD and the RSS. It uses the received information, along with an appropriate search heuristic, to determine the resources that should host the new process. Then, the SA returns the decision to the CL, which, in turn, requests execution of that process through the appropriate MSHN Daemon. The MSHN Daemon invokes the application on its machine.

As a process executes, the CL updates both the RSS and the RRD with the current status of the resources and the requirements of the process. Meanwhile, the SA establishes call-backs with both the RRD and the RSS to notify the SA if either the status of the resources has significantly changed, or the actual resource requirements are substantially different from what was initially returned from the RRD. In either case, if it appears that the assigned resources can no longer deliver the required QoS, the application must be terminated or adapted (e.g., use an alternative implementation that may deliver less QoS, but requires less resources). Upon receipt of a call-back, the SA might require that several of the applications adapt so that more of them can receive their requested QoS.

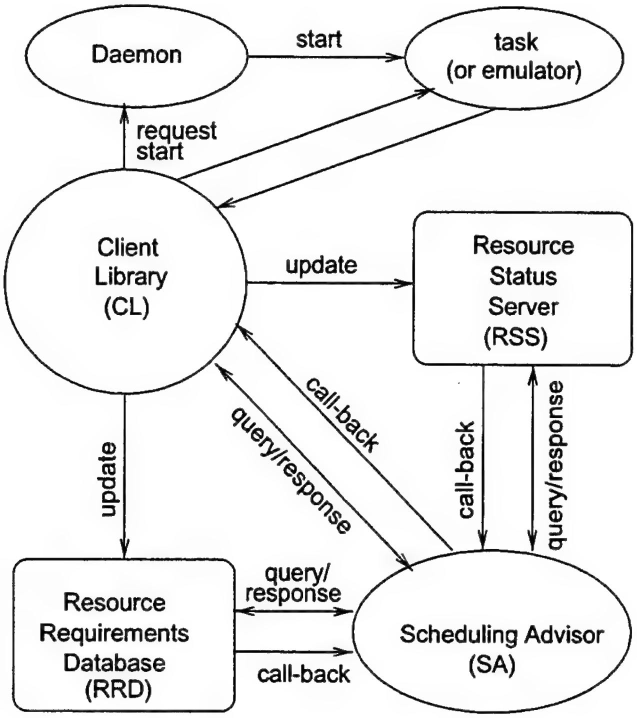


Figure 1: High-level block diagram of the functional architecture of MSHN.

5.3. Research Issues for MSHN's Scheduling Advisor

The formulation of an optimization criterion for mapping tasks in complex HC environments is currently being researched in the HC community. Resource allocation involves attempting to solve heuristically an NP-complete optimization problem. MSHN is developing a criterion that maximizes a weighted sum of values that represents the benefits of delivering the required and desired QoS (including security, priorities, and preferences for versions), within the specified deadlines, if any. MSHN attempts to account for both preferences for various versions and priorities. That is, when it is impossible to deliver all of the most preferred information within the specified deadlines due to insufficient resources, MSHN's optimization criterion is designed to favor delivering the most preferred version to the highest priority applications. In MSHN's optimization criterion, deadlines can be simple or complex. That is, sometimes a user could use a piece of information only if it is received before a specific time. At other times, a user would like to associate a more general benefit function, which would tell how beneficial the information is to user depending on when it is received. Further information about MSHN's optimization criterion can be found in [KiH99].

The relative performance of search algorithms is another research issue. The MSHN team has obtained extensive results identifying the regions of heterogeneity where certain heuristics perform better than others for maximizing throughput by minimizing the time at which the last application, of a set of applications, should complete (e.g., [Arm97, BrS99, MaA99]). Re-targeting of these heuristics to other optimization criteria is currently underway. Additionally, MSHN team members have performed extensive research into accounting for dependencies among subtasks (e.g., [BhP98a, BhP98b, BhP99, WaS98]). The next two sections outline some of the MSHN research in the static and dynamic mapping of meta-tasks in HC environments.

6. Dynamic Heuristics for Mapping Meta-Tasks in HC Systems

6.1. Overview

This section describes and compares eight dynamic heuristics that can be used in an RMS like MSHN for mapping meta-tasks [MaA99]. In an HC system where the tasks to be executed are not known *a priori*, dynamic schemes are necessary to match tasks to machines, and to compute the execution order of the tasks assigned to each machine. A dynamic scheme is also needed in environments where some machines in the suite may go off-line and new machines may come on-line. These dynamic mapping heuristics are non-preemptive, and assume that the tasks have no deadlines or priorities associated with them.

The mapping heuristics can be grouped into two categories: on-line mode and batch-mode heuristics. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The definition of a meta-task can now be refined as the independent set of tasks that is considered for mapping at the mapping events. A meta-task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and ones that were mapped in earlier mapping events but did not begin execution. While on-line mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

The on-line heuristics consider, to varying degrees and in different ways, task affinity for different machines and machine ready times. The batch heuristics consider these factors, as well as aging of tasks waiting to execute. The ready time, r_k, of a machine m_k is defined as the earliest time that machine is going to be ready after completing the execution of the tasks that are currently assigned to it. It is assumed that each time a task t_i completes on a machine m_j a report is sent to the mapper. Because the heuristics presented here are dynamic, the expected machine ready times are based on a combination of actual task execution times and estimated expected task execution times. The experiments conducted in [MaA99] to study dynamic mapping heuristics model this situation using simulated actual values for the execution times of the tasks that have already finished their execution. Also, all heuristics examined in [MaA99] operate in a centralized fashion on a dedicated suite of machines; i.e., the mapper controls the execution of all jobs on all machines in the suite. It is also assumed that the mapping heuristic is being run on a separate machine. The next few subsections condense some discussions and results from [MaA99].

6.2. Background Terms and a Performance Measure for Dynamic Mapping Heuristics

The expected execution time $\underline{e_{ij}}$ of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given m_j has no load when t_i is assigned. The expected completion time $\underline{c_{ij}}$ of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let $\underline{m}$ be the total number of the machines in the HC suite. Let $\underline{K}$ be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be $\underline{a_i}$, and let the begin time of t_i on m_j be $\underline{b_{ij}}$. From the above definitions, $c_{ij} = b_{ij} + e_{ij}$. Let $\underline{c_i}$ be c_{ij}, where machine m_j is the one assigned by the mapping heuristic to execute task t_i. The makespan for the complete schedule is then defined as $max_{t_i \in K}(c_i)$. Makespan is a measure of the throughput of the HC system, and does not measure the quality of service imparted to an individual task. One other performance measure is given in [MaA99].

6.3. On-Line Mode Dynamic Mapping Heuristics

The MCT (minimum completion time) heuristic assigns each task to the machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The MCT is a variation of the fast greedy heuristic from SmartNet [FrG98].

The MET (minimum execution time) heuristic (a variation of the user directed assignment in [FrG98]) assigns each task to the machine that performs that task's computation in the least amount of execution time. This heuristic, in contrast to the MCT, does not consider machine ready times, and can cause a severe imbalance in load across the machines. The main advantage of this method is its simplicity.

The SA (switching algorithm) heuristic [MaA99] is motivated by the following observation. The MET heuristic can potentially create load imbalance across machines by assigning many more tasks to some machines than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. The SA heuristic uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. The purpose is to have a heuristic with the desirable properties of both the MCT and the MET.

Let the maximum ready time over all machines in the suite be r_{max}, and the minimum ready time be r_{min}. Then, the load balance index across the machines is given by $\pi = r_{min}/r_{max}$. The parameter π can have any value in the interval $[0, 1]$. Two threshold values, π_l (low) and π_h (high), for the ratio π are chosen in $[0, 1]$ such that $\pi_l < \pi_h$. Initially, the value of π is set to 0.0. The SA heuristic begins mapping tasks using the MCT heuristic until the value of load balance index increases to at least π_h. After that point in time, the SA heuristic begins using the MET heuristic to perform task mapping. This causes the load balance index to decrease. When it reaches π_l, the SA heuristic cycle continues.

The KPB (k-percent best) heuristic [MaA99] considers only a subset of machines while mapping a task. The subset is formed by picking the $(km/100)$ best machines based on the execution times for the task, where $100/m \leq k \leq 100$. The task is assigned to a machine that provides the earliest completion time in the subset. If $k = 100$, then the KPB heuristic is reduced to the MCT heuristic. If $k = 100/m$, then the KPB heuristic is reduced to the MET heuristic. A "good" value of k maps a task to a machine only within a subset formed from machines computationally superior for that particular task.

The OLB (opportunistic load balancing) heuristic (used for comparisons in [FrG98]) assigns a task to the machine that becomes ready next. It does not consider the execution time of the task when mapping it onto a machine. If multiple machines become ready at the same time, one machine is arbitrarily chosen.

6.4. Batch Mode Dynamic Mapping Heuristics

In the batch mode, meta-tasks are mapped after predefined intervals. For the ith mapping event, the meta-task M_i is mapped at time τ_i, where $i \geq 0$. The initial meta-task, M_0, consists of all the tasks that arrived prior to time τ_0, i.e., $M_0 = \{t_j \mid a_j < \tau_0\}$. The meta-task, M_k, for $k > 0$, consists of tasks that arrived after the last mapping event and the tasks that had been mapped, but did not start executing, i.e., $M_k = \{t_j \mid \tau_{k-1} \leq a_j < \tau_k\} \cup \{t_j \mid a_j < \tau_{k-1}, b_j > \tau_k\}$.

The mapping events may be scheduled using one of the following two strategies. The regular time interval strategy maps the meta-tasks at regular intervals of time except when all machines are busy. When all machines are busy, all scheduled mapping events that precede the one before the expected ready time of the machine that finishes earliest are canceled. The fixed count strategy maps a meta-task M_i as soon as one of the following two mutually exclusive conditions are met: (a) an arriving task makes $\mid M_i \mid$ larger than or equal to a predetermined arbitrary number κ, or (b) all tasks have arrived, and a task completes while the number of tasks which yet have to begin is larger than or equal to κ. In this strategy, the length of the mapping intervals will depend on the arrival rate and the completion rate. The possibility of machines being idle while waiting for the next mapping event will depend on the arrival rate, completion rate, m, and κ.

The Min-min heuristic shown in Figure 2 is based on the ideas presented in [IbK77], and implemented in SmartNet [FrG98]. The Min-min heuristic calculates the minimum completion time for each task in the meta-task currently being considered. The task which has the minimum of these completion times is assigned the corresponding machine, and that machine's ready time is updated. This assigned task is removed from the meta-task and the procedure is repeated.

Min-min begins by scheduling the tasks that change the expected machine ready time status by the least amount that any assignment could. If tasks t_i and t_k are contending for a particular machine m_j, then Min-min assigns m_j to the task (say t_i) that will change the ready time of m_j less. This increases the probability that t_k will still have its earliest completion time on m_j, and shall be assigned to it. Because at $t = 0$, the machine which finishes a task earliest is also the one that executes it fastest, and from thereon the Min-min heuristic changes machine ready time status by the least amount for every assignment, the percentage of tasks assigned their first choice (on basis of expected execution time) is likely to be higher in Min-min than with the other batch mode heuristics described in this subsection. The expectation is that a smaller makespan can be obtained if a larger number of tasks is assigned to the machines that not only complete them earliest but also execute them fastest.

The Max-min heuristic is similar to the Min-min heuristic given in Figure 2. It was one of

```
for all tasks $t_i$ in meta-task $M_v$ (in an arbitrary order)
    for all machines $m_j$ (in a fixed arbitrary order)
        $c_{ij} = e_{ij} + r_j$
do until all tasks in $M_v$ are mapped
    for each task in $M_v$ find its earliest completion
        time and the machine that obtains it
    find the task $t_k$ with the minimum earliest
        completion time
    assign task $t_k$ to the machine $m_l$ that gives the
        earliest completion time
    delete task $t_k$ from $M_v$
    update $r_l$
    update $c_{il}$ for all $t_i$ in $M_v$
enddo
```

Figure 2: The Min-min heuristic.

the heuristics implemented in SmartNet [FrG98]. Once the machine that provides the earliest completion time is found for every task, the task t_k that has the maximum earliest completion time is determined and then assigned to the corresponding machine. The matrix c and vector r are updated and the above process is repeated with tasks that have not yet been assigned a machine.

The Max-min is likely to do better than the Min-min heuristic in the cases where there are many more shorter tasks than the long tasks. For example, if there is only one long task, Max-min will execute many short tasks concurrently with the long task. The resulting makespan might just be determined by the execution time of the long task in these cases. Min-min, however, first finishes the shorter tasks (which may be more or less evenly distributed over the machines) and then executes the long task, increasing the makespan compared to the Max-min.

The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it [MaA99]. Figure 3 shows the Sufferage heuristic. Let the sufferage value of a task t_i be the difference between its second earliest completion time (on some machine m_y) and its earliest completion time (on some machine m_x). That is, using m_x will result in the best completion time for t_i, and using m_y the second best. The Sufferage heuristic attempts to assign each task based on the MCT. When there is a conflict for a machine, the task with the higher sufferage value is assigned to that machine.

```
for all tasks $t_k$ in meta-task $M_v$ (in an arbitrary order)
    for all machines $m_j$ (in a fixed arbitrary order)
        $c_{kj} = e_{kj} + r_j$
do until all tasks in $M_v$ are mapped
    mark all machines as unassigned
    for each task $t_k$ in $M_v$ (in an arbitrary order)
        find machine $m_j$ that gives the earliest
            completion time
        sufferage value = second earliest completion
            time − earliest completion time
        if machine $m_j$ is unassigned
            assign $t_k$ to machine $m_j$, delete $t_k$
            from $M_v$, mark $m_j$ assigned
        else
            if sufferage value of task $t_i$ already
            assigned to $m_j$ is less than the
            sufferage value of task $t_k$
                unassign $t_i$, add $t_i$ back to $M_v$,
                assign $t_k$ to machine $m_j$,
                delete $t_k$ from $M_v$
    endfor
    update the vector $r$ for the tasks that
        were assigned to the machines
    update the $c$ matrix
enddo
```

Figure 3: The Sufferage heuristic.

6.5. Sample Comparisons for Dynamic Mapping Heuristics

For many heuristics, there are control parameter values and/or control function specifications that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and/or information in the literature. The above is also true for the static mapping heuristics presented in the next section. In Figures 4 and 5, vertical lines at the tops of bars show minimum and maximum values for the 50 trials, while the bars show the averages.

In Figure 4, the on-line mode heuristics are compared based on makespan for inconsistent HiHi heterogeneity. The KPB provides the minimum makespan, closely followed by the MCT.

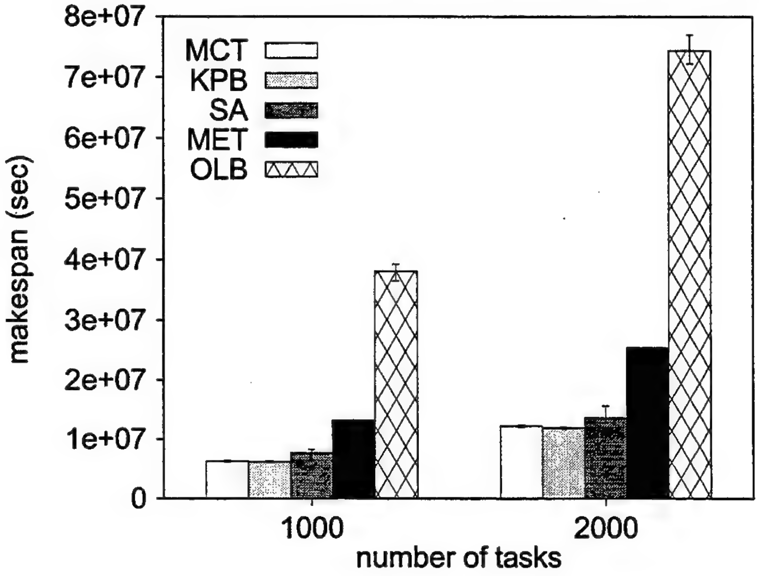


Figure 4: Makespan for the on-line heuristics for inconsistent HiHi heterogeneity, using 20 machines, based on 50 trials.

Figure 5 shows the makespan for batch heuristics under similar conditions. The Sufferage heuristic gives the smallest makespan, followed by the Min-min. When the task arrival rate is relatively higher, the batch method outperformed the on-line method in these studies. The reader is referred to [MaA99] for a detailed description of the experiments, further analysis, and more results.

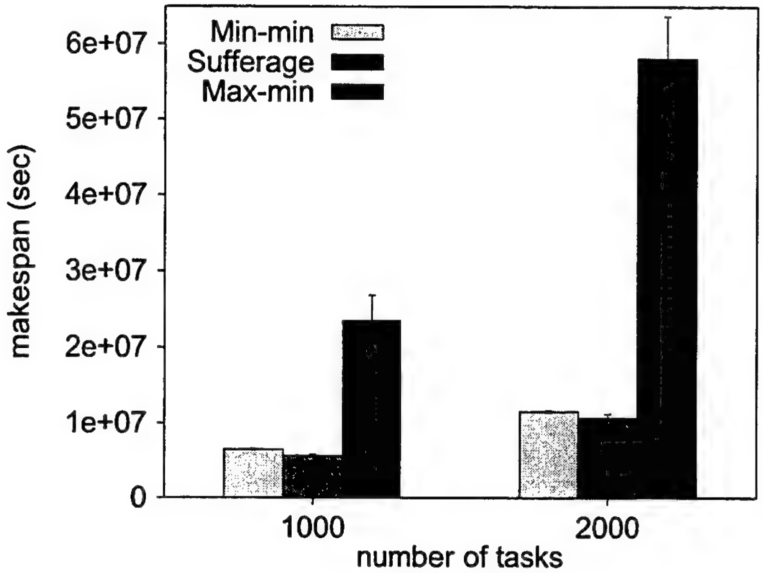


Figure 5: Makespan of the batch heuristics for the regular time interval strategy and inconsistent HiHi heterogeneity, using 20 machines, based on 50 trials.

7. Static Heuristics for Mapping Meta-Tasks in HC Systems

7.1. Overview

This section describes and compares eleven static heuristics that can be used in an RMS like MSHN for mapping meta-tasks to machines. In a general HC system, static mapping schemes are likely to make better mapping decisions because more time can be devoted for the computation of schedules off-line than on-line. However, static schemes require that the set of tasks to be mapped be known *a priori*, and that the estimates of expected execution times of all tasks on all machines be known with reasonable accuracy. A meta-task, in the context of static heuristics, is the set of all independent tasks that are being considered for mapping. Like the dynamic heuristics in the previous section, these static mapping heuristics are non-preemptive, assume that the tasks have no deadlines or priorities associated with them, and assume a dedicated HC system.

7.2. Description of Static Heuristics

This subsection consists of brief definitions of the eleven static meta-task mapping heuristics that are studied and fully described in [BrS99]. The basic terms and the performance measure defined for the dynamic heuristics in Sections 6.1 and 6.2 hold for static heuristics as well, except for the terms that characterize the dynamic nature of the dynamic heuristics, e.g., fixed count strategy.

The descriptions below assume that the machine ready times are updated after each task is mapped. For cases when tasks can be considered in an arbitrary order, the order in which the tasks appeared in the ETC matrix was used.

The static OLB (opportunistic load balancing) heuristic is similar to its dynamic counterpart except that it assigns tasks in an arbitrary order, instead of order of arrival. The UDA (user directed assignment) heuristic [ArH98] works in the same way as the MET heuristic except that it maps tasks in an arbitrary order instead of order of arrival. The fast greedy heuristic [ArH98] is the same as the MCT, except that it maps tasks in an arbitrary order instead of their order of arrival. The static Min-min heuristic works in the same way as the dynamic Min-min, except a meta-task contains all the tasks in the system. The static Max-min heuristic works in the same way as the dynamic Max-min, except a meta-task has all the tasks in the system. The greedy heuristic performs both the static Min-min and static Max-min heuristics, and uses the better solution [ArH98, FrG98].

The GA (genetic algorithm) is a popular technique used for searching large solution spaces. The version of the heuristic used for this study was adapted from [WaS98] for this particular HC environment. Figure 6 shows the steps in a general genetic algorithm [SrP94].

```
initial population generation;
evaluation;
while (stopping criteria not met)
    selection;
    crossover;
    mutation;
    evaluation;
endwhile
```

Figure 6: General procedure for a genetic algorithm.

The genetic algorithm implemented here operates on a population of 200 chromosomes (possible mappings) for a given meta-task. Each chromosome is a $|\ K\ |$ vector, where position i $(0 \leq i < t)$ is the machine to which the task t_i has been mapped. The initial population is generated using two methods: (a) 200 chromosomes randomly generated from a uniform distribution, or (b) one chromosome that is the Min-min solution and 199 random chromosomes. The latter method employs the seeding of the population with a Min-min chromosome. In this implementation, the GA executes eight times (four times with initial populations from each method), and the best of the eight mappings is used as the final solution. The makespan serves as the fitness value for evaluation of the evolution.

SA (simulated annealing) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This solution uses the same representation for a solution as the chromosome for the GA. SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [RuN95]). This probability is based on a system temperature that decreases for each iteration. As the system temperature "cools," it is more difficult for currently poorer solutions to be accepted.

The GSA (genetic simulated annealing) heuristic is a combination of the GA and SA techniques [ShW96]. In general, GSA follows procedures similar to the GA outlined above. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simplified SA decision process for accepting or rejecting new chromosomes.

The Tabu search keeps track of the regions of the solution space which have already been searched so as not to repeat a search near these "Tabu" areas [GlL97]. A solution (mapping) uses the same representation as a chromosome in the GA approach. Heuristic searches are conducted within a region, and the best solution for that region is stored. Then, a new region, not on the tabu list, is searched. When a stopping criterion is reached, the best solution among regions is selected.

The final heuristic in the comparison study is known as the A* heuristic. A* is a tree-based search that has been applied to many other task allocation problems (e.g., [ChL91, RuN95]). The technique used here is similar to the one described in [ChL91]. As the tree grows, intermediate nodes represent partial solutions (a subset of tasks are assigned to machines), and leaf nodes represent final solutions (all tasks are assigned to machines). The partial solution of a child node has one more task $\underline{t_a}$ mapped than the parent node. Each parent node can be replaced by its m children, one for each possible mapping of t_a. The number of nodes allowed in the tree is bounded to limit mapper execution time. Less promising nodes are deleted, and the more promising nodes are expanded. The process continues until a leaf node (complete mapping) is reached.

7.3. Sample Comparisons for Static Mapping Heuristics

Figures 7 and 8 show comparisons of the eleven static heuristics using makespan as the criterion in two different heterogeneity environments. Vertical lines at the top of bars show minimum and maximum values for the 100 trials, while the bars show the averages. It can be seen that, for the parameters used in this study, GA gives the smallest makespan for both inconsistent HiHi and inconsistent HiLo heterogeneities. The reader is referred to [BrS99] for more results, details, and discussions.

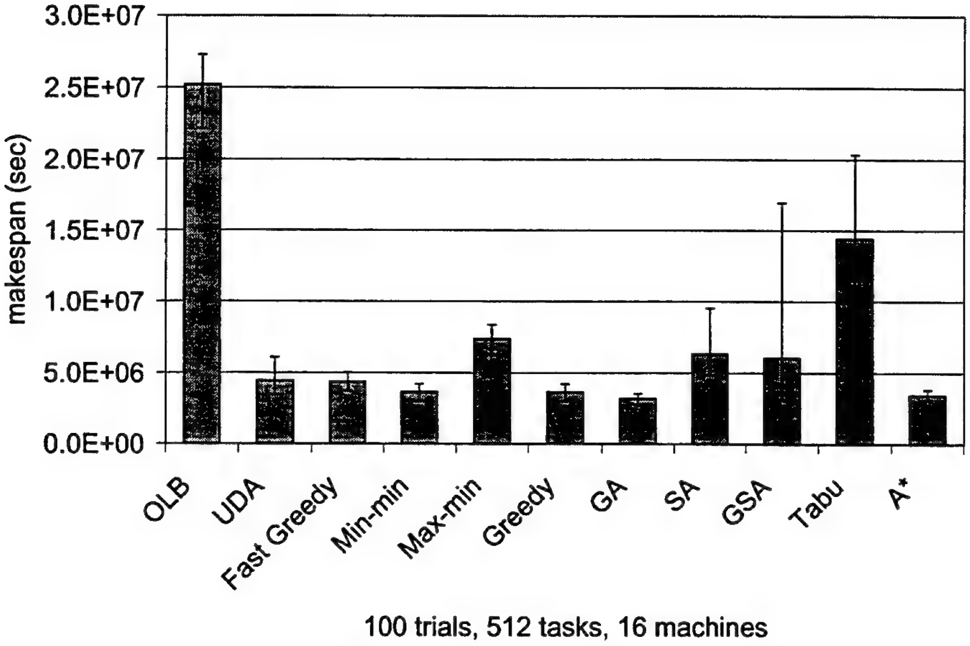


Figure 7: Inconsistent, high task, high machine heterogeneity.

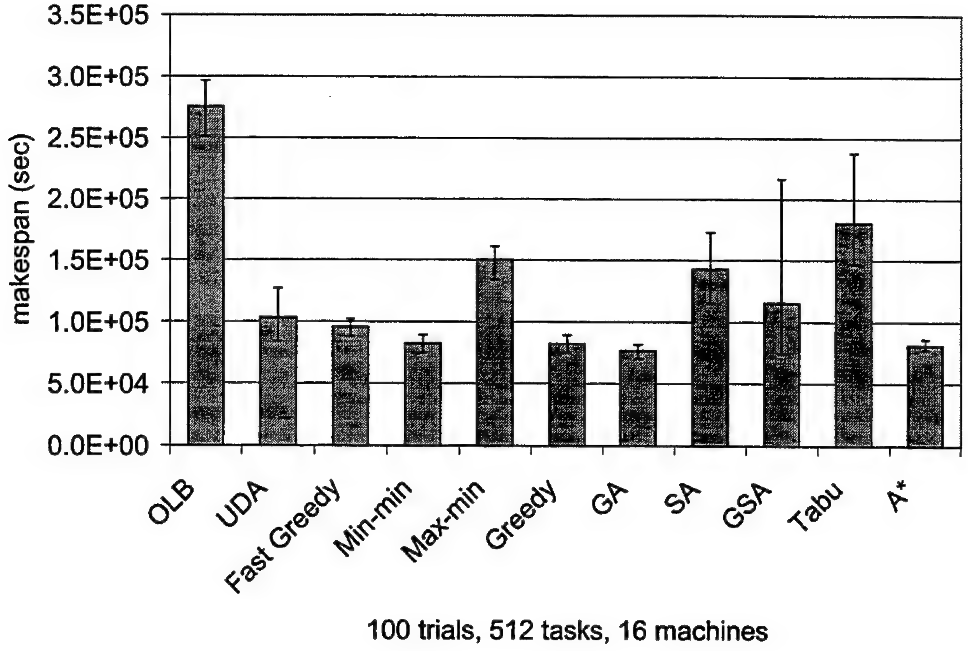


Figure 8: Inconsistent, high task, low machine heterogeneity.

8. Conclusions

Heteroegeneous computing is a relatively new research area for the computer field. Interest in such systems continues to grow, both in the research community and in the user community.

Some of the different types of HC systems that have been built were discussed here, including mixed-mode, multi-mode, and mixed-machine. Mixed-machine HC was then focussed upon. A way to describe different kinds of heterogeneous environments based on characteristics of the ETC matrix was presented. The structure of a taxonomy for heuristics for mapping tasks onto mixed-machine HC systems was reviewed. As an example of the design of a resource management system for such HC environments, the high-level functional architecture of MSH-N was provided. Finally, a sampling of heuristic techniques for dynamic and static mapping of independent tasks onto machines in an HC suite was given. The definition of each of these heuristics was summarized, and some example comparison results were shown. For all of these topics, references were cited where much greater detail can be found. The reader is encouraged to pursue these references to learn more.

Acknowledgments: The authors thank Tracy Braun and Surjamukhi Chatterjea for their valuable comments.

References

[ArH98] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predications," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79–87.

[Arm97] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance*, Thesis, Department of Computer Science, Naval Postgraduate School, 1997 (D. Hensgen, Advisor).

[BhP98a] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, "Adaptive communication algorithms for distributed heterogeneous systems," *IEEE International Symposium on High Performance Distributed Computing*, July 1998, pp. 310–321.

[BhP98b] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, "Block-cyclic redistribution over heterogeneous networks," *International Conference on Parallel and Distributed Computing Systems*, Sep. 1998, pp. 242–249.

[BhP99] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra, "Efficient collective communication in distributed heterogeneous systems," *IEEE International Conference on Distributed Computing Systems*, June 1999, to appear.

[BrS98] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *1998 IEEE Symposium on Reliable Distributed Systems*, Oct. 1998, pp. 330–335.

[BrS99] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, R. F. Freund, and D. Hensgen, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 15-29.

[ChL91] K. Chow and B. Liu, "On mapping signal processing algorithms to a heterogeneous multiprocessor system," *1991 International Conference on Acoustics, Speech, and Signal Processing - ICASSP 91*, Vol. 3, 1991, pp. 1585–1588.

[Esh96] M. M. Eshaghian (ed.), *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[Fer89] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[FiC91] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting," *Journal of Parallel and Distributed Computing*, Vol. 11, No. 3, Mar. 1991, pp. 239–251.

[FoK99] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Fransisco, CA, 1999.

[FrG98] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 184–199.

[FrK96] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for metacomputing," *2nd International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN '96)*, June 1996, pp. 514–521.

[Fre89] R. F. Freund, "Optimal selection theory for superconcurrency," *Supercomputing '89*, Nov. 1989, pp. 699–703.

[GhY93] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[GlL97] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, MA, 1997.

[HeK99] D. A. Hensgen, T. Kidd, D. St. John, M. C. Schnaidt, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R. F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: The Management System for Heterogeneous Networks," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 184–198.

[IbK77] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18–27.

[KiH99] J.-K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. St. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, *A Multi-Dimensional QoS Performance Measure for Distributed Heterogeneous Networks*, Technical Report, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in preparation, 1999.

[MaA99] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 30–44.

[MaB99] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," in *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., John Wiley, New York, NY, 1999, to appear.

[ReT85] R. van Renesse and A. S. Tanenbaum, "Distributed operating systems," *ACM Computing Surveys*, Vol. 17, No. 4, Dec. 1985, pp. 419–470.

[RuN95] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[ShW96] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 98–117.

[SiM96] H. J. Siegel, M. Maheswaran, D. W. Watson, J. K. Antonio, and M. J. Atallah, "Mixed-mode system heterogeneous computing," in *Heterogeneous Computing*, M. M. Eshaghian, ed., Artech House, Norwood, MA, 1996, pp. 19–65.

[SrP94] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17–26.

[WaS98] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 1–15.

[WeL89] C. C. Weems, S. Levitan, A. R. Hanson, E. M. Riseman, and J. G. Nash, "The image understanding architecture," *International Journal of Computer Vision*, Vol. 2, No. 3, Jan. 1989, pp. 251–282.

Biographies

Howard Jay Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received BS degrees in both electrical engineering and management from MIT, and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Shoukat Ali is pursuing an MSEE degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a Research Assistant. His main research topic is dynamic mapping of meta-tasks in heterogeneous computing systems. He has held teaching positions at Aitchison College and Keynesian Institute of Management and Sciences, both in Lahore, Pakistan. He was also a Teaching Assistant at Purdue. Shoukat received his BS degree in electrical and electronic engineering from the University of Engineering and Technology, Lahore, Pakistan in 1996. His research interests include computer architecture, parallel computing, and heterogeneous computing.

Mapping Tasks onto Heterogeneous Computing Systems

Howard Jay Siegel and Muthucumaru Maheswaran
{hj@purdue.edu, maheswar@ecn.purdue.edu}
Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285, USA

Abstract – The goal of this invited tutorial paper is to provide an overview of three current research efforts in heterogeneous computing that focus on methods for determining a mapping of an application onto a heterogeneous suite of machines. The first study involves a genetic-algorithm approach for mapping the subtasks of an application task onto the machines in a distributed heterogeneous system. This is a static compile-time approach that must be used off-line (prior to task execution) due to its long run time. The second topic is the high-level design of components of an intelligent operating system for mapping and dynamically remapping automatic target recognition tasks onto a heterogeneous parallel platform. The intelligent operating system uses a new technique for dynamically selecting new mappings on-line during task execution from among choices precomputed off-line. Last, some initial preliminary results from a new research project for designing a dynamic mapping heuristic that does not use precomputed mappings is described. This dynamic heuristic is fast and is suitable for operation during application execution. Future research directions are discussed for all three projects.

Keywords: automatic target recognition, distributed computing, genetic algorithms, heterogeneous computing, mapping, matching, MSHN, parallel processing, scheduling.

1. Introduction

High-performance computers typically achieve only a fraction of their peak capabilities on certain portions of some application tasks. This is because different subtasks of an application can have very different computational requirements that result in needs for different machine capabilities. A single machine architecture cannot satisfy all the computational requirements in certain applications equally well. Thus, the use of a heterogeneous computing environment is more appropriate.

The research presented in Section 2 was supported by NRaD under subcontract number 20-950001-70; the research presented in Section 3 was supported by the Department of Defense, Small Business Innovative (SBIR) Program, funded by the Army Research Laboratory, under contract number DAAL01-96-C-0031; and the research presented in Section 4 is supported by the DARPA/ITO Quorum Program under NPS subcontract number N62271-97-M-0900.

One type of heterogeneous computing (HC) system provides a variety of architectural capabilities, orchestrated to perform an application whose subtasks have diverse execution requirements [SiA96]. A mixed–machine system is a heterogeneous suite of different types of independent machines interconnected by a high-speed network. When the goal is to perform an application task, each subtask must be matched to the machine that results in the lowest overall task execution time. To exploit HC systems in such situations, a task must be decomposed into subtasks, where each subtask is computationally homogeneous, and different subtasks may have different machine architectural requirements. These subtasks may share initial or generated data, creating the potential for inter-machine data transfer overhead.

Examples of applications that have been implemented on mixed-machine HC systems include a simulator for mixing in turbulent convection [KlM93], the interactive rendering of multiple earth science data sets on the CASA testbed [Spe90], and the computation of the radiation treatment planning for cancer patients on VISTAnet [RoC92]. Typically, users must perform task decomposition and subtask matching and scheduling. When performing matching and scheduling, the user must consider machine availability, the way in which each subtask's computational requirements match each machine's capabilities, inter-machine shared data transfers, and others factors [SiA96]. One long-term pursuit in HC is to do decomposition, matching, and scheduling automatically [SiD97].

This invited tutorial paper provides an overview of three current research efforts in distributed and parallel mixed-machine HC. In particular, this tutorial paper focuses on examples of methods for determining a mapping of an application onto a heterogeneous suite of machines (i.e., a matching of application subtasks to machines and a scheduling for the execution order and inter-machine communications of these subtasks). A brief summary of each project is given, including references for more complete information if available. Possible future research directions for each project are also given. A broader view of the HC field and related open research problems can be found in [Esh96, FrS93, KhP93, SiA96, SiD97].

The first study, presented in Section 2, involves a genetic-algorithm approach developed at Purdue University for mapping the subtasks of an application task onto the machines in a distributed heterogeneous system [WaS98]. This is a static (i.e., compile-time) approach that must be used off-line (prior to task execution) due to its long run time.

In some situations, the execution characteristics of an application task may not be derivable prior to run time and changes during run time as a function of the input data. When this occurs, a single static mapping may not be effective over a long time period, and dynamic mapping and remapping schemes should be considered, such as the two approaches discussed next.

The second topic, explored jointly by Purdue University and Architecture Technology Corporation, is the high-level design of components of an intelligent operating system for mapping and dynamically remapping automatic target recognition tasks onto a heterogeneous parallel platform [BR97a, BR97b]. The intelligent operating system uses a new technique for dynamically selecting new mappings on line during task execution from among choices precomputed off line. Section 3 provides an overview of this work.

Section 4 describes some initial preliminary results from a new Purdue research project for designing a dynamic mapping heuristic that does not use precomputed mappings. This dynamic heuristic is fast and is suitable for operation during application execution. It is based on a list-scheduling type of algorithm.

The goal of this invited tutorial paper is to introduce the reader to these three current approaches to mapping research in the field of HC. The reader is encouraged to learn more about them, and other research topics in HC, from the references cited. Furthermore, the reader is encouraged to consider examining the future research problems listed in the following sections and in [KhP93, SiA96, SiD97, Sun92].

2. A Genetic-Algorithm Approach for Task Mapping

In general, the problem of performing matching and scheduling in an HC environment is NP-complete [Fer89], and therefore some heuristic must be employed. As an example of current HC research on mapping statically (i.e., at compile time), a genetic-algorithm approach from [WaS98] is summarized. An application task is decomposed into a set of subtasks $\underline{S}$ of size $\underline{|S|}$. Let $\underline{s_i}$ be the i-th subtask. An HC suite consists of a set of machines $\underline{M}$. Let $\underline{m_j}$ be the j-th machine. Each machine can be a different type. The global data items are data items that need to be transferred between subtasks.

The following assumptions about the applications and HC environment are made. Each application task will be represented by a DAG (directed acyclic graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the the data dependencies between subtasks. (Note that while the subtasks' dependencies are represented as a DAG, subtasks themselves may contain loops.) For each global data item, there is a single subtask that produces it (producer) and there are some subtasks that need this data item (consumers). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred over it. This task has exclusive use of the HC environment, and the genetic-algorithm-based mapper controls the HC machine suite (hardware platform). Subtask execution is non-preemptive. The estimated expected execution time of each subtask on each machine is known. For each pair of machines in the HC suite, an equation for estimating the time to send data between those machines as a function of data set size is known.

Genetic algorithms (GAs) provide a promising heuristic approach to optimization problems that are intractable [Dav91, Gol89, Hol75, RiT94, SrP94]. The first step is to encode some of the possible solutions as chromosomes, the set of which is referred to as a population. In the [WaS98] approach, each chromosome consists of two parts: the matching string and the scheduling string. Let mat be the matching string, which is a vector of length $|S|$. If $\text{mat}(i) = j$, then subtask s_i is assigned to machine m_j. Typically, multiple subtasks will be assigned to the same machine, and then executed in a non-preemptive manner based on an ordering that obeys the precedence constraints (data dependencies) specified in the task DAG. The scheduling string is a topological sort of the DAG representing the task (i.e., a valid total ordering of the partially ordered DAG). Define ss to be the scheduling string, which is a vector of length $|S|$. If $\text{ss}(k) = i$, then subtask s_i is the k-th subtask in the total ordering. Because it is a topological sort, if subtask $\text{ss}(k)$ is a consumer of a global data item produced by subtask $\text{ss}(j)$, then $j < k$. The scheduling string gives an ordering of subtasks that is used by the evaluation step.

Each chromosome is associated with a fitness value, which is the completion time of the solution (i.e., mapping) represented by this chromosome (i.e., the expected execution time of the application task if the mapping specified by this chromosome were used). Overlapping among all of the computations and communications performed is limited only by inter-subtask data dependencies and the availability of the machines and the inter-machine network.

In the initial population generation step, a predefined number of distinct chromosomes are randomly created. The solution from a non-evolutionary heuristic is also included in the initial population. After the initial population is determined, the genetic algorithm iterates until a predefined stopping criterion is met. Each iteration consists of the selection, crossover, mutation, and evaluation steps.

In the selection step, some members of the population are removed and others are duplicated. First, all of the chromosomes in the population are ordered (ranked) by their fitness values. Then a rank-based roulette wheel selection scheme is used to implement proportionate selection [Hol75]. The population size is kept constant and a chromosome representing a better solution has a higher probability of having one or more copies in the next generation population. This GA approach also incorporates elitism, i.e., the best solution found so far is always maintained in the population [Rud94].

The selection step is followed by the crossover step, where some chromosomes are paired and corresponding components of the paired chromosomes are exchanged. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings in the current population. For each pair, it randomly generates a cutoff point, and divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in the bottom part of one string is the relative positions of these subtasks in the other original scheduling string, thus guaranteeing that the newly generated scheduling strings are valid schedules. The crossover operator for the matching strings randomly chooses some pairs of the matching strings in the current population. For each pair, it randomly generates a cutoff point, and divides both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The next step is mutation. The scheduling string mutation operator randomly chooses some scheduling strings in the current population. Then for each chosen scheduling string, it randomly selects a victim subtask. The valid range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed and still have a valid topological sort of the DAG. The victim subtask is moved randomly to another position in the scheduling string within its valid range. The matching string mutation operator randomly chooses some matching strings in the current population. For each chosen matching string, it randomly selects a subtask entry. Then the machine assignment for the selected entry is changed randomly to another machine.

The last step of an evolution iteration is the evaluation step to determine the fitness value of each chromosome in the current population. A communication subsystem that is modeled after a HiPPI LAN with a central crossbar switch [ToR93] was assumed for the tests that were conducted. As stated earlier, the above steps of selection, crossover, mutation, and evaluation are repeated until one of the stopping criteria are met: (1) the number of iterations reaches some limit (e.g., 1000), (2) the population converged (all the chromosomes had the same fitness value), or (3) the best solution found was not improved after some number of iterations (e.g., 150).

In the tests of this GA approach in [WaS98], simulated program behaviors were used. Small-scale tests were conducted with up to ten subtasks, three machines, and population size 50. For each test, the GA approach found a solution (mapping) that had the same expected completion time for the task as that of the optimal solution found by exhaustive search. Larger tests with up to 100 subtasks, 20 machines, and population size 200 were conducted. This GA approach produced solutions (mappings) that averaged from 150% to 200% better than those produced by the non-evolutionary levelized min-time (LMT) heuristic proposed in [IvO95]. The heuristic in [IvO95] was selected for comparison because it used a similar model of HC. The GA approach took much more time to generate the mappings than did the LMT approach; however, if the mappings are being done at compile time for production tasks that will be executed repeatedly, the generation time is worthwhile.

There are number of ways this GA approach for HC task mapping may be built upon for future research. These include extending this approach to allow multiple producers for each of the global data items, parallelizing the GA approach, developing evaluation procedures for other communication subsystems, and considering loop and data-conditional constructs that involve multiple subtasks. This last extension will be particularly difficult to handle at compile time if the loop bounds and data-conditional constructs are input-data dependent.

3. Dynamic Use of Statically-Derived Mappings

This section, which provides a simplified summary of [BR97a], focuses on a particular application domain (iterative automatic target recognition (ATR) tasks) and an associated specific class of dedicated HC hardware platforms. The contribution of [BR97a] is that, for the computational environment considered, it presents a methodology for on-line (i.e., execution-time) input-data dependent remapping of the application subtasks to the processors in the HC hardware platform using one of a set of previously stored off-line (i.e., statically) determined mappings. That is, the operating system will be able to decide during the execution of the application whether or not to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system will be able to select a previously derived and stored mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking).

This high-level operating system approach for enabling the on-line use of off-line mappings is called the IOS (Intelligent Operating System). The ATR Kernel component makes decisions on how a given ATR application task should be performed, including determining the partial ordering of subtasks and which algorithms should be used to accomplish each subtask [BR97b]. The HC Kernel component decides how the partially ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. Also, the HC Kernel interacts with the Basic Kernel component (i.e., the low level operating system) to execute the application and monitor its execution. Thus, the ATR Kernel deals with application issues, while the HC Kernel deals with implementation issues. Information from the Algorithm Database and the Knowledge

Base is used to support the ATR and HC Kernels.

The IOS has not been implemented; such an implementation is a major undertaking and outside the scope of [BR97a], which is the design concepts for the HC Kernel. The IOS design has its roots in the high-level model presented in [ChD89] for automatic dynamic processor allocation in a partitionable parallel machine with homogeneous processors. The IOS differs from other real-time HC mapping techniques in that it allows on-line execution-time use of off-line precomputed mappings. This is significant because off-line heuristics can produce better mappings as a result of much longer execution times to search for a good solution than what is practical for an on-line heuristic. Thus, the mapping quality of a time-consuming off-line heuristic can be approached at real-time speeds.

This work is being developed for a class of ATR applications each of which can be modeled as an iterative execution of a set of partially ordered subtasks, i.e., applications that will iteratively execute a DAG such as described in Section 2. The expected number of subtasks is ten to 50. It is assumed that each ATR application in the class under consideration is a production job that is executed repeatedly. Thus, it is worthwhile to invest off-line time in preparing an effective mapping of the application onto the hardware platform used to execute it. The ATA (automatic target acquisition) system described in [DaB90] is an example of such an iterative ATR application.

The type of target hardware platforms considered for this study are driven by the expected needs of the kinds of ATR applications that are of interest to the U.S. Army Research Laboratory (e.g., [DaE94]). For the intended application environment, it is assumed that there will be up to four different types of processors (e.g., SHARC, PowerPC), and up to a total of 64 processors (of all types combined). These processors will comprise the heterogeneous parallel architecture onto which application tasks will be mapped. The IOS approach is appropriate for larger HC platforms as well.

For the initial iteration through the set of subtasks, the IOS will employ user-provided information about the processing environment in its selection of algorithms for the subtasks, and their associated implementations. As part of this, the IOS will choose a precomputed mapping to decide how to assign one or more processors to each subtask.

Dynamic parameters are characteristics of the given application that may change during run time and can be computed by the application as it executes, as a function of the input data. The values of dynamic parameters may influence the decision of how to map a task onto the HC platform, and even initiate the use of a different algorithm for a given subtask. Examples of dynamic parameters include: (1) amount of clutter found in the current image of a sequence and (2) number of objects currently located that need to be identified. The application developer is expected to define a small set of dynamic parameters that will have the most impact on the execution time of the subtasks in the application.

After each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image stream being processed. It is expected that the values of these parameters will change slowly. After all subtasks have completed execution for a given iteration, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel. The HC Kernel will use the most recent values of such dynamic parameters to estimate if it is worthwhile to reconfigure the assignment of processing resources to subtasks to reduce the execution time of the next iteration. If it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks. If not, the same assignment will continue to be used.

To provide the HC kernel with the decision capability and off-line mappings it will need, the IOS has a Scenario Generator, which is the component of the IOS that uses information provided by the application developer to derive representative values for these dynamic parameters. Assume $\underline{D}$ dynamic parameters are selected for a given application. Let the number of representative choices for the i-th dynamic parameter be $\underline{C_i}$, $0 \le i < D$. Each set of D values for these D dynamic parameters, one per parameter, is called a scenario. The number of different scenarios that can be generated is $\underline{\sigma} = C_0 \times C_1 \times \cdots \times C_{D-1}$. It is important to pick an effective set of dynamic parameters and associated representative choices for parameter values, and to keep $|\sigma|$ at a reasonable size. For each scenario, an off-line heuristic, such as a version of the GA described in Section 2, is used to generate a mapping that is stored in a D-dimensional array in the Knowledge Base and indexed by the dynamic parameter values for that scenario. The expected execution time of the task for that given scenario and mapping is also stored.

As the application is executing, the HC Kernel monitors the run-time values of the dynamic parameters at the end of each iteration through the underlying DAG to decide whether to continue with the current mapping, or to select and implement a new mapping (for the next iteration) from among the off-line generated mappings. It does this by finding the scenario whose associated dynamic parameter values are closest to the actual dynamic parameter values at the end of the iteration. It then compares the previously stored execution time for the new mapping associated with that scenario to perform an iteration of the DAG versus the actual time for the last iteration plus an estimated reconfiguration time. Thus, the off-line processing provides a set of predetermined mappings that the on-line processing can index in real time.

The IOS ideas can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the iterative ATR applications and platforms considered here. Examples of other such application domains include sensor-based robotics, intelligent vehicle highway systems, air traffic control, nuclear facility maintenance, weather prediction, intruder detection, and manufacturing inspection.

Future work based on this IOS involves actual implementation of the components discussed and a graphical user interface. The Algorithm Database and Knowledge Base must be populated with the information needed for the IOS to be a functioning ATR system for some set of applications. Implementation of the IOS will require the study of various research problems, such as: how to select the number of scenarios to use in a given environment; how to facilitate the selection of dynamic parameters and their representative values; how to effectively merge multiple ATR tasks to execute simultaneously on the same HC platform and be dynamically remapped as required by the IOS (even when the iteration times for the tasks are different); and how to reasonably estimate the remapping reconfiguration overhead time as a function of the application and platform. Another area for investigation is how to use this technique in broader environments where the dynamic parameters correspond to general computational properties rather than being a function of the application.

4. Dynamically-Derived Mappings

Preliminary results from a new research project on a dynamic remapping heuristic for HC systems is summarized in this section. The heuristic described here uses current system information (e.g., machine loading at run time) to improve a an initial static mapping by periodic dynamic remapping. As in Sections 2 and 3, it is assumed that the subtasks of the application(s) are represented by a DAG and the dynamic-heuristic-based mapper controls the HC machine suite. The dynamic mapper executes on a dedicated workstation that is tightly coupled with the HC machine suite. The scheduler allows multiple applications to execute concurrently on the HC machine suite, i.e., at any given time the subtasks executing on the system can be from different applications.

The dynamic heuristic executes in two phases. In the first phase, a priority is computed for each subtask from the static mapping that is provided to the algorithm (i.e., any static mapping can be used). During this phase, which is executed at compile time, the DAG is level sorted into m level sets, numbered consecutively from 0 to $m-1$. The level sorting clusters the subtasks into level sets such that the subtasks within a level set are independent (i.e., there are no direct data dependencies among the subtasks within a level). Furthermore, for the j-th subtask at level k, $\underline{s_{j,k}}$, there exists at least one incident edge (data dependency) such that the source subtask is at level $k-1$, i.e., an incident edge from some $s_{i,k-1}$. The level set at level $m-1$ includes only those subtasks without any descendents and at level 0 includes only those subtasks without any predecessors.

The second phase executes in real time with the execution of the subtasks. In this phase, dynamic remappings are performed if such remappings are expected to yield a performance benefit. The dynamic remappings are non-preemptive and they involve updating the mapping for the next level before the execution of that level begins.

The priority assignment begins with the level set at level $m-1$ and assigns a priority to each subtask within the level set. The priority of each subtask in the $(m-1)$-th level set is its expected computation time on the machine it was assigned by the static matching. Now consider the k-th level, $0 \leq k < m-1$. Let $\underline{e_i}$ be the expected computation time of the subtask $s_{i,k}$ for the given static matching, $\underline{c_{ij}}$ be the communication time for a descendent $s_{j,q}$ of $s_{i,k}$ to get all the relevant data items from $s_{i,k}$ (where $q \geq k+1$), $\underline{\text{priority}(s_{i,k})}$ be the priority of the subtask $s_{i,k}$, and $\underline{\text{set of descendents}}$ of $s_{i,k}$ be the set of subtasks in level(s) $q \geq k+1$ such that each subtask is dependent on a data item generated by subtask $s_{i,k}$. With the above definitions, the priority of a subtask $s_{i,k}$ is given by:

$$\text{priority}(s_{i,k}) = e_i + \max_{s_{j,q} \in \textit{set of decendents}} (c_{ij} + \text{priority}(s_{j,q})).$$

The priorities of the subtasks in other level sets can be computed using a similar recursion. The priority of a subtask can be interpreted as the length of the critical path from the point the given subtask is located on the DAG to the end of the execution of all its descendents. The dynamic remapping heuristic is based on the idea that by executing the subtasks with higher priorities as fast as possible it is possible to expect a shorter completion time for the application.

The execution of the subtasks of an application proceeds from level 0 to level $m-1$. Consider the situation where the dynamic heuristic is trying to remap the subtasks at the i-th level while the subtasks at the $(i-1)$-th level are being executed. The dynamic heuristic starts remapping the i-th level subtasks when the first $(i-1)$-th level subtask begins its execution. When level i is being scheduled, it is highly likely that actual execution time information can be used for most subtasks from levels 0 to $i-2$. There may be some long-running subtasks from levels 0 to $i-2$ that could be still executing when subtasks from level i are being considered for remapping. For such subtasks expected execution times are used. The dynamic heuristic examines the subtasks in the i-th level set in descending order based on their precomputed static priorities. The subtask is assigned to the machine that gives the shortest partial completion time for the subtasks that have been scheduled (including this subtask).

A simulator was implemented to evaluate the performance of the remapping heuristic versus a static mapper called the baseline heuristic [WaS98]. The baseline heuristic initially uses level sorting to levelize the subtasks. Then all subtasks are ordered such that a subtask at level i comes before one at level j, $i < j$. The subtasks in the same level are arranged in descending order based on the number of descendents of each subtask (ties broken arbitrarily). The subtasks are considered for assignment in this order. The subtask is assigned to the machine that gives the shortest partial completion time for the subtasks that have been scheduled (including this subtask).

The inputs to the simulator are the randomly generated DAG, the static mapping, the percentage deviation in the parameters (subtask execution times or communication times), and the machine and network information. The percentage deviation is defined as the percentage by which the simulated expected time differs from the simulated actual time. If $\underline{\alpha}$ is the percentage deviation and $\underline{u}$ a random number that is uniformly distributed in [0,1], then the simulated actual value of a parameter x can be modeled as $x(100-\alpha+2\alpha u)/100$. The simulator evaluates the execution of the static mapping by computing the completion times of the subtasks using the simulated actual parameter values and the static mapping. The remapped execution is simulated by computing the completion times for the simulated actual parameter values and the dynamically remapped mapping.

The simulation results are shown in Figure 1 for ten machines and 100 subtasks. Each data point is an average of ten simulation runs. The average simulation time for each run of this experiment was 0.62 seconds and the average time for executing the remapping heuristic per level of the task graph was 0.0041 seconds, with a minimum of 0.00312 seconds and a maximum of 0.00764 seconds. As the percentage deviation of the parameters from their static estimates increases, the difference between the task completion time using the ‘‘static mapping only’’ versus the task completion time using the ‘‘dynamic remapping’’ increases. Thus, dynamic remapping can improve the performance of an initial static mapping. However, the time taken by the dynamic heuristic to remap the i-th level of the task graph must be less than the time difference between the time a $(i-1)$-th level subtask started execution and the time the machine and data necessary to start the execution of an i-th level subtask becomes available; otherwise, the dynamic heuristic may delay the execution of an i-th level subtask.

Other groups have also studied dynamic mapping heuristics for HC systems (e.g., [FrC97, HaL95, LeP95]). Methods presented in these papers are different from the work presented here. Future work involves implementing their algorithms in the simulator discussed here to compare the performance of the dynamic heuristic presented here with those presented in the above papers.

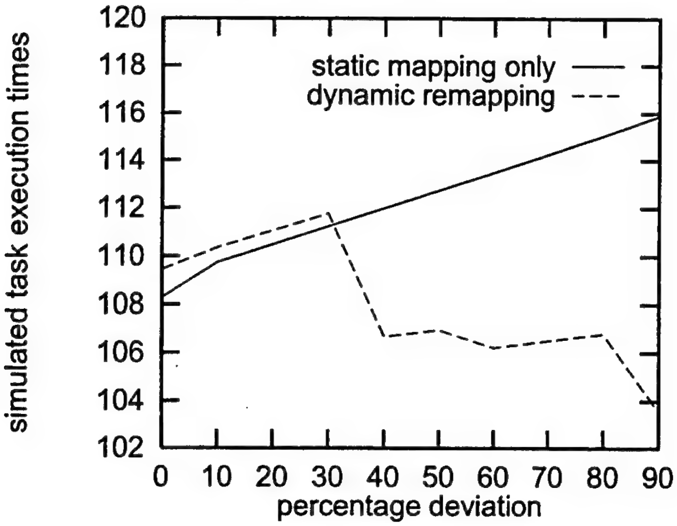


Figure 1: Simulation results for the dynamic heuristic for ten machines and 100 subtasks.

The following are some areas for future research for the dynamic heuristic. More simulation data should be collected, e.g., 100 test runs per data point compared to the ten runs used for the initial study. Simulation studies should be conducted to compare the performance of a dynamic remapper with a static GA-based mapper (such as described in Section 2) under varying parameter values. In particular, because the GA-based mapper has been shown to provide mappings superior to the much faster baseline heuristic when the expected execution times are used as actual times [WaS98], it will be interesting to examine at what percentage variation the dynamic remapping makes a significant improvement over the GA (this can be done using the baseline or the GA as the initial static mapper). Other areas include investigating the use of alternate level set definitions and priority computation schemes, and exploring ways of improving the dynamic heuristic to support multiple data-copies (a situation where a subtask can have multiple sources (machines) for a needed data item) [TaS97].

5. Conclusions

A novel genetic-algorithm approach for task matching and scheduling in HC environments was presented in Section 2 [WaS98]. It is applicable to the static scheduling of production jobs and can be readily used for scheduling multiple independent tasks (and their subtasks) collectively. For small-scale tests, the proposed approach found optimal solutions. For larger tests, it outperformed a published non-evolutionary heuristic.

In Section 3, for the computational environment considered, an HC Kernel was described for making real-time on-line input-data-dependent remappings of the application subtasks to the processors in the HC hardware platform using previously stored off-line statically determined mappings (e.g., using a GA) [BR97a]. The IOS ideas can also be used for other application domains and other HC hardware platforms whose characteristics are similar to those considered there.

A different approach to dynamic remapping was discussed in Section 4. Here the actual mapping for one set of subtasks is computed concurrently with the execution of an earlier set of subtasks. When a subtask is mapped, any available run-time information about the system state is used by the mapping heuristic. This research is just beginning, and the initial results are very promising.

Some of the future research outlined at the ends of Sections 2 to 4 may be pursued as part of a DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort that includes NPS (Naval Postgraduate School), NRaD (a Naval Laboratory), Purdue, and USC (University of Southern California). It builds on SmartNet, an operational scheduling framework and system for managing resources in a heterogeneous environment developed at NRaD [FrK96]. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

HC is an exciting research field with practical uses. The reader in encouraged to investigate the research problems listed at the ends of Sections 2 to 4 and to explore this field further using the references cited.

Acknowledgments – The authors thank Janet M. Siegel for her comments on this paper. A version of some of this material also appeared in an invited keynote paper for the 1997 Parallel and Distributed Processing Techniques and Applications Conference.

References

[BR97a] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "On-line use of off-line derived mappings for iterative automatic target recognition tasks and a particular class of hardware platforms," *6th Heterogeneous Computing Workshop (HCW '97)*, Apr. 1997, pp. 96-110.

[BR97b] J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "Modeling ATR applications for intelligent execution upon a heterogeneous computing platform," *1997 IEEE Int'l Conf. Systems, Man, and Cybernetics (SMC '97),* Oct. 1997, to appear.

[ChD89] C. H. Chu, E. J. Delp, L. H. Jamieson, H. J. Siegel, F. J. Weil, and A. B. Whinston, ''A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture,'' *J. of Parallel and Distributed Computing,* Vol. 6, No. 3, June 1989, pp. 598-622.

[DaB90] P. David, S. Balakirsky, and D. Hillis, ''A real-time automatic target acquisition system,'' *Conf. on Unmanned Vehicle Systems,* July 1990, pp. 183-198.

[DaE94] P. David, P. Emmerman, and S. Ho, ''A scalable architecture system for automatic target recognition,'' *13th AIAA/IEEE Digital Avionics Systems Conf.,* Oct. 1994, pp. 414-420.

[Dav91] L. Davis, ed., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, NY, 1991.

[Esh96] M. M. Eshaghian, ed., *Heterogeneous Computing,* Artech House, Norwood, MA, 1996.

[Fer89] D. Fernandez-Baca, ''Allocating modules to processors in a distributed system,'' *IEEE Trans. on Software Engineering,* Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.

[FrC97] R. F. Freund, B. R. Carter, D. Watson, E. Keith, F. Mirabile, and H. J. Siegel, ''Generational scheduling for heterogeneous computing systems,'' *Information Science Journal,* Special Issue on Parallel and Distributed Processing Techniques and Applications, accepted and scheduled to appear in 1997.

[FrK96] R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, ''SmartNet: A scheduling framework for meta-computing,'' *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96),* June 1996, pp. 514-521.

[FrS93] R. F. Freund and H. J. Siegel, ''Heterogeneous processing,'' *IEEE Computer,* Special Issue on Heterogeneous Processing, Vol. 26, No. 6, June 1993, pp. 13-17.

[Gol89] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, Reading, MA, 1989.

[HaL95] B. Hamidzadeh, D. J. Lilja, and Y. Atif, ''Dynamic scheduling techniques for heterogeneous computing systems,'' *Concurrency: Practice and Experience,* Vol. 7, No. 7, Oct. 1995, pp. 633-652.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems,* Univ. of Michigan Press, Ann Arbor, MI, 1975.

[IvO95] M. A. Iverson, F. Ozguner, and G. J. Follen, ''Parallelizing existing applications in a distributed heterogeneous environment,'' *4th Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 93-100.

[KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, ''Heterogeneous computing: Challenges and opportunities,'' *IEEE Computer,* Vol. 26, No. 6, June 1993, pp. 18-27.

[KlM93] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, ''A case study in metacomputing: Distributed simulations of mixing in turbulent convection,'' *2nd Workshop on Heterogeneous Processing (WHP '93),* Apr. 1993, pp. 101-106.

[LeP95] C. Leangsuksun, J. Potter, and S. Scott, ''Dynamic task mapping algorithms for a distributed heterogeneous computing environment,'' *4th Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 30-34.

[RiT94] J. L. Ribeiro Filho and P. C. Treleaven, ''Genetic-algorithm programming environments,'' *IEEE Computer,* Vol. 27, No. 6, June 1994, pp. 28-43.

[RoC92] J. Rosenman and T. Cullip, ''High-performance computing in radiation cancer treatment,'' *CRC Critical Reviews in Biomedical Engineering,* Vol. 20, 1992, pp. 391-402.

[Rud94] G. Rudolph, ''Convergence analysis of canonical genetic algorithms,'' *IEEE Trans. Neural Networks,* Vol. 5, No. 1, Jan. 1994, pp. 96-101.

[SiA96] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, ''Heterogeneous computing,'' in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio, ''Software support for heterogeneous computing,'' in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[Spe90] ''Special report: Gigabit network testbeds,'' *IEEE Computer*, Vol. 23, No. 9, Sep. 1990, pp. 77-80.

[SrP94] M. Srinivas and L. M. Patnaik, ''Genetic algorithms: A survey,'' *IEEE Computer,* Vol. 27, No. 6, June 1994, pp. 17-26.

[Sun92] V. S. Sunderam, ''Design issues in heterogeneous network computing,'' *Workshop on Heterogeneous Processing (WHP '92),* revised edition, Mar. 1992, pp. 101-112.

[TaS97] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, ''Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system,'' *IEEE Trans. on Parallel and Distributed Systems*, accepted and scheduled to appear in 1997.

[ToR93] D. Tolmie and J. Renwick, ''HiPPI: Simplicity yields success,'' *IEEE Network,* Vol. 7, No. 1, Jan. 1993, pp. 28-32.

[WaS98] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. ''Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach,'' *J. of Parallel and Distributed Computing,* Special Issue on Parallel Evolutionary Computing, accepted and scheduled to appear in 1998. (A preliminary version appeared in the *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 72-85.)

Pre-copy-edited version of paper that appeared in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 11. Nov. 1998, pp. 1088-1101.

A Stochastic Model for Heterogeneous Computing and Its Application in Data Relocation Scheme Development

Min Tan [*] and *Howard Jay Siegel* [†]

[*] Segue Software Inc.
142 South Santa Cruz Ave.
Los Gatos, CA 95030
mtan@segue.com

[†] Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285, USA
hj@purdue.edu

Submitted in March 1997
Revised in June 1998

Abstract

In a dedicated mixed-machine heterogeneous computing (HC) system, an application program may be decomposed into subtasks, then each subtask assigned to the machine where it is best suited for execution. Data relocation is defined as selecting the sources for needed data items. It is assumed that multiple independent subtasks of an application program can be executed concurrently on different machines whenever possible. A theoretical stochastic model for HC is proposed, in which the computation times of subtasks and communication times for inter-machine data transfers can be random variables. The optimization problem for finding the optimal matching, scheduling, and data relocation schemes to minimize the total execution time of an application program is defined based on this stochastic HC model. The global optimization criterion and search space for the above optimization problem are described. It is validated that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. The local optimization criterion established by the greedy approach, coupled with the search space defined for choosing valid data relocation schemes, can help developers of future practical data relocation heuristics.

Keywords: data relocation, greedy algorithm, heterogeneous computing, mapping, matching, optimization, scheduling, stochastic modeling.

This research was supported in part by the DARPA/ITO Quorum Program under NPS subcontract number N62271-98-M-0217, and by NRaD

1: Introduction

A single application program often requires many different types of computation that result in different needs for machine capabilities. Heterogeneous computing (HC) is the effective use of the diverse hardware and software components in a heterogeneous suite of machines connected by a high-speed network to meet the varied computational requirements of a given application [CiL95, FrS93, KhP93, SiA96, SiD97, ZhY95]. One goal of HC is to decompose an application program into subtasks, each of which is computationally homogeneous, and then assign each subtask to the machine where it is best suited for execution.

Subtask matching, scheduling, and data relocation are three critical steps for implementing an HC application on an HC system. Matching involves assigning subtasks to machines. Scheduling includes ordering the execution of the subtasks assigned to each machine and ordering the inter-machine communication steps for data transfers. Data relocation is the scheme for selecting the sources for needed data items. Here, a stochastic HC model is developed and used as a basis to study theoretical issues for data relocation. The practical implication of the theoretical results derived on data relocation heuristic design is explained. It is assumed that multiple independent subtasks of an application program can be executed concurrently on different machines whenever possible (e.g., when the machines are available for subtask execution).

The contribution of this paper can be summarized as follows. A theoretical stochastic HC model is proposed, in which the computation times of subtasks and communication times for inter-machine data transfers are modeled as random variables. The rest of this paper focuses on theoretical issues for data relocation using a stochastic HC model. The optimization problem for finding the optimal matching, scheduling, and data relocation schemes to minimize the total execution time of an application program executed in a dedicated HC system is defined based on this proposed stochastic HC model. The global optimization criterion and search space for the above optimization problem in HC are described. It is validated that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics in practice. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. The local optimization criterion established by the greedy approach, coupled with the search space defined for choosing valid data relocation schemes, can help developers of future practical data relocation heuristics.

The inter-machine communication time between subtasks can be substantial and is one of the major factors that degrade the performance of an HC system. This paper focuses on potential methods for minimizing the inter-machine communication time of an application program when the concurrent execution of different subtasks on different machines is considered whenever possible. In particular, the impact of the data relocation scheme on the total execution time of the subtasks executed in a dedicated HC system is examined.

In most of the mathematical models for HC in the literature (e.g., [ChE93, IvO95, NaY94, TaA95, TaS97, WaK92]), the computation times and inter-machine data transfer times of data items for different subtasks in the application program are assumed to be deterministic quantities. This is valid when the inter-machine network is completely controlled by the scheduler and all execution times and inter-machine communication needs are known *a priori* (not dependent on input data). However, there are elements of uncertainty (e.g., input-data-dependent conditional and looping constructs) that impact the deterministic nature of both the computation and inter-machine communication times for different subtasks. Such uncertainties can create others, e.g., network contention among different inter-machine data transfer steps. They are unpredictable prior to execution time. One approach to modeling these computation and communication times is to represent them as random variables with assumed probability distribution functions.

To use a dedicated HC system to execute an application program efficiently, the optimization problem of using matching, scheduling, and data relocation schemes to minimize the total execution time must be defined. Section 2 provides the background and terminology needed for the rest of this paper. In Section 3, a theoretical stochastic HC model for matching, scheduling, and data relocation is introduced. Based on the random variables of the HC model and given matching, scheduling, and date relocation schemes, a procedure for determining the execution time of an application program (with partially ordered subtasks) is presented in Section 4. In Section 5, a method is devised to enumerate all the valid options in choosing the data relocation scheme for a given arbitrary matching. The cases in which the application programs may include inter-subtask conditional and inter-subtask looping constructs are considered. Thus, Sections 3, 4, and 5 collectively define the above optimization problem in HC with a stochastic model. Because of the complexity of this defined optimization problem in HC, guidelines for devising heuristics must be provided. It is validated in Section 6 that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. This theoretical result indicates that a greedy algorithm based approach can achieve reasonable local optimization for developing data relocation heuristics in practice.

Most of the literature for HC has concentrated on addressing the practical aspects and heuristics for matching and scheduling. This paper emphasizes instead the theoretical issues involved in data relocation using a stochastic HC model. The practical implication of the theoretical results derived on data relocation heuristic design is explained.

This research was supported in part by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks). MSHN is a collaborative research effort among NPS (Naval Postgraduate School), Noemix (a company specializing in software technology for distributed computing), Purdue University, and USC (University of Southern California). It builds on SmartNet [FrG98], an operational scheduling framework and system for managing resources in a heterogeneous environment developed at the NRaD naval laboratory, which also supported this research. The technical objective of MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested qualities of service.

2: Background and Terminology

The material in this subsection is summarized from [TaS97]. It provides the background and terminology needed for the rest of this paper. In general, the goal for HC is to assign each subtask to one of the machines in the system such that the total execution time (computation time and inter-machine communication time) of the application program is minimized [ChE93, Fre89, NaY94, WaK92]. The subtask to machine assignment problem is referred to as matching in HC. When a subset of subtasks can be executed in any order, varying the order of the computation of these subtasks (while maintaining the data dependencies among all subtasks) can impact the total execution time of the application program. Determining the order of computation for the subtasks is referred to as scheduling in HC. In most of the literature for HC, a subtask flow graph is used to describe the data dependencies among subtasks in an application program (e.g., [IvO95, NaY94, SiL93, TaA95, TaS97, Tow86]). In Figure 1, each vertex of the subtask flow graph represents a subtask. Let $S[k]$ denote the k-th subtask. For each data element that $S[k]$ transfers to $S[j]$ during execution, there is an edge from $S[k]$ to $S[j]$ labeled with the corresponding variable name. An extra vertex labeled *Source* denotes the locations where the initial data elements of the program are stored.

Let a data item be a block of information that can be transferred between subtasks. Using information from the subtask flow graph, a data item is denoted by the two-tuple (s, d), where $s \geq 0$ is the number of the subtask that generates the needed value of variable d upon completion of computation of that subtask. If the needed value of d is an initial data element to the program, then $s = -1$. Two data items are the same if and only if they are both associated with the same variable name in an application program and the corresponding value of the data is generated by the same subtask (which implies that the two data items have the same value).

In general, most of the graph-based algorithms for matching-related problems assume that the pattern of data transfers among subtasks is known *a priori* and can be illustrated using a subtask flow graph (e.g., [IvO95, NaY94, SiL93, Tow86]). Thus, no matter which machine is used

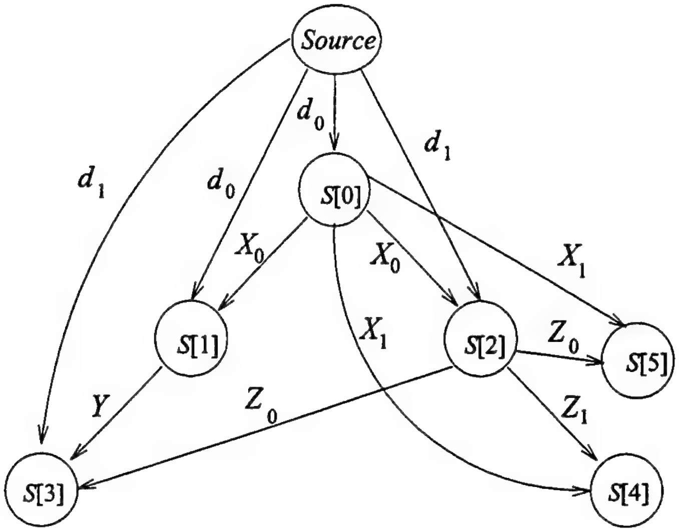


Figure 1: Subtask flow graph for the example application program.

for executing each subtask of a specific application program, the locations (subtasks) from which each subtask obtains its corresponding input data items are determined by the subtask flow graph and are independent of any particular matching scheme between machines and subtasks.

The above assumption generally needs refinement in the case of HC. In [TaS97], two data-distribution situations, namely data locality and multiple data-copies, are identified for addressing refinements of the above assumption. It is assumed that each subtask $S[i]$ keeps a copy of each of its individual input data items and output data items on the machine to which $S[i]$ is assigned by the matching scheme. Furthermore, it is also assumed that all input data items are received for a subtask prior to that subtask's computation.

Data locality arises when two subtasks, $S[j]$ and $S[k]$ that are assigned to the same machine, need the same data item e from $S[i]$ (assigned to a different machine). Because a machine can fetch a data item from its local storage faster than fetching it from other machines, if $S[j]$ is executed after $S[k]$, then $S[j]$ should obtain e locally from $S[k]$ instead of from the machine assigned to $S[i]$. If a subtask flow graph is used to compute inter-subtask communication cost, then without considering machine assignments, the impact of data locality might be ignored.

The multiple data–copies situation arises when two subtasks, $S[j]$ and $S[k]$, need the same data item e from $S[i]$, where $S[i]$, $S[j]$, and $S[k]$ are assigned to three different machines. If $S[k]$ is executed after $S[j]$ obtains e, then the machine assigned to $S[k]$ can get data item e from either the machine assigned to $S[i]$ or the machine assigned to $S[j]$. The choice that results in the shorter time should be selected. Selecting the sources for needed data items is referred to as data relocation (because the data relocation scheme determines the source machines from which the data items will be relocated to the destination machines). In general, when using information only from the subtask flow graph, the possibility of having multiple sources for a needed data item is not considered. Data locality can be viewed as a special case of having multiple data copies (i.e., one copy is on the machine to which the receiving subtask is assigned by the matching scheme).

In [TaS97], it is assumed that, at any instant in time during the execution of an application program, only one inter-machine data transfer step is being executed. All computation and inter-machine communication times of subtasks are assumed to be known deterministic quantities, and any data conditional and looping constructs must be contained within a single subtask. Based on these assumptions, a minimum spanning tree based algorithm is presented in [TaS97] that finds, for a given matching, the optimal scheduling scheme for inter-machine data transfer steps and the optimal data relocation scheme for each subtask. Data locality and multiple data-copies are all considered in the above algorithm. The mathematical model for HC presented in this paper differs from the one in [TaS97] in that, here, limited only by inter-subtask data-dependencies and machine assignments, at any instant in time, multiple subtasks can be executed and multiple inter-machine data transfers can be performed. Also, here the computation times of subtasks and communication times for inter-machine data transfers can be random variables. Furthermore, the cases in which the application programs may include inter-subtask conditional and inter-subtask looping constructs are considered. Thus, the HC model presented here is much more general than the one in [TaS97], which makes the data relocation more complex. It is validated in this paper that a greedy algorithm based approach can establish a local optimization criterion for developing data relocation heuristics. This result indicates that a greedy algorithm based approach can achieve reasonable local optimization for developing data relocation heuristics in practice.

3: A Stochastic Model for Matching, Scheduling, and Data Relocation in HC

A stochastic model of matching, scheduling, and data relocation for HC is formalized in this section. This model is an extension of the one,presented in [TaS97]. The possible concurrent execution of both the computation of subtasks and inter-machine communication steps in an application program is considered. The issues related to using a theoretical stochastic HC model are addressed. When the computation time of each subtask on each machine and the communication times of transferring data items have stochastic properties, those timing parameters must be modeled as random variables. This paper examines underlying theoretical issues with respect to data relocation. Due to the theoretical nature of the proof of the main result in this paper, it is not necessary to know the actual distribution functions of those random variables. The mathematical model presented in this section allows the material in the rest of this paper to be given in unambiguous terms. All notation developed in the remaining sections is summarized in the appendix for the glossary of notation at the end of this paper.

(1) An application program $\underline{P}$ is composed of a set of $\underline{n}$ subtasks

$$\underline{S} = \{S[0], S[1], ..., S[n-1]\}.$$

There are a set of $\underline{Q}$ initial data elements

$$\{d_0, d_1, ..., d_{Q-1}\}.$$

(2) Suppose that $\underline{NI[i]}$ is the number of input data items required by $S[i]$ and $\underline{NG[i]}$ is the number of output data items generated by $S[i]$. There are two sets of data items associated with each $S[i]$. One is the input data set

$$\underline{I[i]} = \{Id[i, 0], Id[i, 1], ..., Id[i, NI[i] - 1]\},$$

the other is the generated output data set

$$\underline{G[i]} = \{Gd[i, 0], Gd[i, 1], ..., Gd[i, NG[i] - 1]\}.$$

The program structure of P is specified by a subtask flow graph.

In this paper, the subtask flow graph of any application program P is assumed to be acyclic. A cycle in a graph represents a loop containing one or more subtasks. With the presence of the inter-subtask looping constructs, an appropriate statistical approach can be used to determine the distribution for the number of iterations each looping construct will execute and the maximum number of iterations each looping construct has [Tow86]. Then, the existent subtask flow graph can be transformed into an acyclic one by unrolling each looping construct with the known or estimated maximum number of iterations. This is the approach presented in Subsection 5.3.2. The above approach potentially will increase the number of subtasks present in the acyclic subtask flow graph significantly. Also, the distribution for the number of iterations each looping construct will execute and the maximum number of iterations each looping construct has can be difficult to estimate in reality. A possibly more practical approach is to group a fixed number of consecutive iterations of each unrolled looping construct together as a single subtask to decrease the number of subtasks present. Another approach is to view each looping construct as part of a single subtask and the boundaries for decomposing an application program into subtasks are not allowed to be in the middle of a looping construct.

(3) An HC system consists of a heterogeneous suite of $\underline{m}$ machines

$$\underline{M} = \{M[0], M[1], ..., M[m - 1]\}.$$

M includes the devices where all the initial data elements are stored before the execution of the application program P.

(4) There is a <u>computation matrix</u> $\underline{C} = \{C[i, j]\}$, where $\underline{C[i, j]}$ denotes the computation time of $S[i]$ on machine $M[j]$ (e.g. [GhY93, YaK94]). For the reason stated in Section 1, $C[i, j]$ is assumed to be a random variable with a known distribution. It can be computed from empirical information or by applying two characterization techniques in HC, namely task profiling and analytical benchmarking (see [SiA96] for a survey of these techniques). In [LiA95], a methodology is introduced for estimating the distribution of execution time for a given data parallel program that is to be executed on a single hybrid SIMD/SPMD mixed-mode machine. This methodology is extended in [LiA97] for estimating the distribution of execution time for an application program that is to be executed on a mixed-machine HC system. However, as mentioned earlier, for the results mentioned here, it is not necessary to determine the distribution functions for the random variables.

(5) The matching associated with the application program P is defined by an assignment function $\underline{Af}$: $S \rightarrow M$ such that if $Af(i) = j$, then $S[i]$ is assigned to be executed on machine $M[j]$. $\underline{NS[j]}$ is defined as the number of subtasks assigned to be executed on machine $M[j]$. Thus, $\sum_{j=0}^{m-1} NS[j] = n$.

(6) A scheduling function $\underline{Sf}$ indicates the execution order of a subtask with respect to the other subtasks assigned to the same machine. If $Sf(i) = k$, then $S[i]$ is the k-th subtask whose computation is executed on machine $M[Af(i)]$, where $0 \le k < NS[Af(i)]$. Readers should notice that the scheduling function Sf schedules only the order of the computation for different subtasks (*not* the order for executing the inter-machine communication steps).

(7) The set of <u>data–source</u> <u>functions</u> is

$$\underline{DS} = \{DS[0], DS[1], ..., DS[n-1]\},$$

where $DS[i](j) = [k_1, k_2]$ $(0 \le i < n, 0 \le j < NI[i], 0 \le k_1 < n$, and $0 \le k_2 < m)$ means that $S[i]$ obtains the input data item $Id[i, j]$ from $S[k_1]$ and $k_2 = Af(k_1)$. If $DS[i](j) = [k_1, k_2]$ and $k_1 = -1$, then $Id[i, j] = (-1, d_x)$ and $S[i]$ obtains the associated initial data element from machine $M[k_2]$ where d_x is initially stored. Readers should notice that, when $k_1 \ne -1$, the augmented information k_2 can be obtained with the known Af and is redundant. But the information from k_2 is necessary to specify the source of an initial data element when $k_1 = -1$. The above definition of *DS* gives a unified way of specifying the values of a data-source function. If each subtask fetches its input data items only from the sources where they are generated (in the case of the initial data elements, from their initial locations), there exists only one choice of *DS* for each specific *Af* and *Sf*. But if the impact of the data locality and multiple data-copies is considered, there are different choices for *DS*. This choice of *DS* corresponds to the data relocation problem discussed in Section 2.

It is assumed that each subtask *S*[*i*] will submit a copy of its input data item *Id*[*i*, *j*] to the network for forwarding to other destination machines (based on *DS*) immediately after *Id*[*i*, *j*] is available on machine *M*[*Af*(*i*)]. Each subtask will also submit copies of all of its output data items to the network to be transferred to the proper destination machines (based on *DS*) after the completion of its entire computation. Thus, *Af*, *Sf*, and *DS* together completely specify the computation and inter-machine communication steps needed at any time to execute the application program *P* in a dedicated HC system.

(8) The communication time estimator *D*[*s*, *r*, *e*] denotes the length of the communication time interval between the time when a data item *e* is available on *M*[*s*] and the time when *e* is obtained by *M*[*r*] (assuming this transfer is required for the given *Af*, *Sf*, and *DS*). For the reason stated in Section 1, *D*[*s*, *r*, *e*] is assumed to be a random variable (again recall that the distribution of this random variable is not needed to derive the results of this paper). *D*[*s*, *r*, *e*] includes all the various hardware and software related times of the inter-machine communication process (e.g., network latency and the time for data format conversion between *M*[*s*] and *M*[*r*] when necessary).

Most of the literature for HC (e.g., [GhY93, KhP92, TaA95, TaS97]) assumes that the inter-machine communication time for sending a data item *e* from *M*[*s*] to *M*[*r*] is only a function of *s*, *r*, and *e*. But in reality, even in a dedicated HC system, when an application program is executed, the traffic pattern for inter-machine communication can be impacted by subtask computation and other inter-machine communication times that are all input data dependent (and represented as random variables). The choice of *Af*, *Sf*, and *DS* impacts all of these computation and communication times and, hence, the communication time interval between the time when *e* is available on *M*[*s*] and the time when *e* is obtained by *M*[*r*]. Thus, the communication time estimator *D*[*s*, *r*, *e*] is dependent on *Af*, *Sf*, *DS*, *s*, *r*, and *e*.

In general, it will be extremely difficult (if not impossible) to estimate the distribution function of *D*[*s*, *r*, *e*] as a function of *Af*, *Sf*, *DS*, *s*, *r*, and *e*. The purpose of defining *D*[*s*, *r*, *e*] here is to address the factors that impact the inter-machine communication times for the application programs executed in a dedicated HC system. It also helps to establish a theoretical model for defining the global optimization criterion of the optimization problem for HC. With this well-defined theoretical model and global optimization criterion, the greedy algorithm based approach introduced in Section 6 can provide potential data relocation heuristics with a sound local optimization criterion based on a solid theoretical derivation. Within the matching and scheduling problem domain, many researchers have shown that local optimization is a worthwhile approach to achieve global optimization (e.g., [ElL90, IvO95, SiL93]). Thus, future data relocation heuristics can follow the local optimization criterion in Section 6 to achieve a reasonable level of global optimization without the information about the exact distribution function of *D*[*s*, *r*, *e*].

4: A Topological Sort Based Algorithm for Calculating the Execution Time of an Application Program in an HC System

In this section, a topological sort based algorithm for calculating the total execution time (computation and communication times) of an HC application program is introduced. This algorithm helps establish the global optimization criterion of the optimization problem for HC with respect to matching, scheduling, and data relocation.

For a given computation matrix *C* and communication time estimator *D*[*s*, *r*, *e*], the total execution time of the application program *P* associated with an assignment function *Af*, a scheduling function *Sf*, and a set of data-source functions *DS* is defined by the following procedure. A data relocation graph (denoted as *Gr*) corresponding to a particular *Af*, *Sf*, and *DS* is generated using the steps specified below. When the impact of data locality and multiple data-copies is considered, the concept of a valid set of data-source functions *DS* of the application program *P* can be defined according to the properties of *Gr*. There may be many valid sets for *P*, each corresponding to a unique graph for *P*, and each resulting in possibly different execution time of *P*. An invalid *DS* would correspond to a set of data-source functions that does not result in an operational program.

The steps for constructing *Gr* are as follows.

Step 1: A *Source* vertex is generated that represents the locations of all the initial data elements (which may be on different machines).

Step 2: For each *S*[*i*], *NI*[*i*] + 1 vertices are created, one for each of the *NI*[*i*] input data items and one for all of the generated output data items of *S*[*i*]. These are the set of input data vertices, labeled *V*[*i*, *j*] ($0 \leq j < NI[i]$) and the output data vertex $V_g[i]$ (as shown in Figure 2). *V*[*i*, *j*] represents the operation for subtask *S*[*i*] to receive its *j*-th input data item. $V_g[i]$ represents the computation for *S*[*i*] to generate all of its output data items. *V* is a set that contains all of the above vertices associated with the application program *P* in Steps 1 and 2. Each *V*[*i*, *j*] is associated with a weight zero and each $V_g[i]$ is associated with a weight *C*[*i*, *Af*(*i*)], the computation time of subtask *S*[*i*] on the machine assigned by the assignment function *Af*.

Step 3: For any input data vertex $V[i_1, j_1]$, suppose that $DS[i_1](j_1) = [i_2, k_2]$ where $-1 \leq i_2 < n$ and $0 \leq k_2 < m$, and if $0 \leq i_2 < n$, then $k_2 = Af(i_2)$.

Case A: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from the *Source* vertex if $Id[i_1, j_1] = (-1, d_k)$ and d_k is one of the initial data elements.

If $i_2 = -1$, then there exists k ($0 \leq k < Q$), such that $Id[i_1, j_1] = (-1, d_k)$, and a directed edge with weight $D[k_2, Af(i_1), Id[i_1, j_1]]$ is added from the *Source* vertex to $V[i_1, j_1]$ (recall that $DS[i_1](j_1) = [i_2, k_2]$ implies that d_k is received from machine $M[k_2]$). That is, if subtask $S[i_1]$'s j_1-th input data item $Id[i_1, j_1]$ is one of the initial data elements and is obtained from

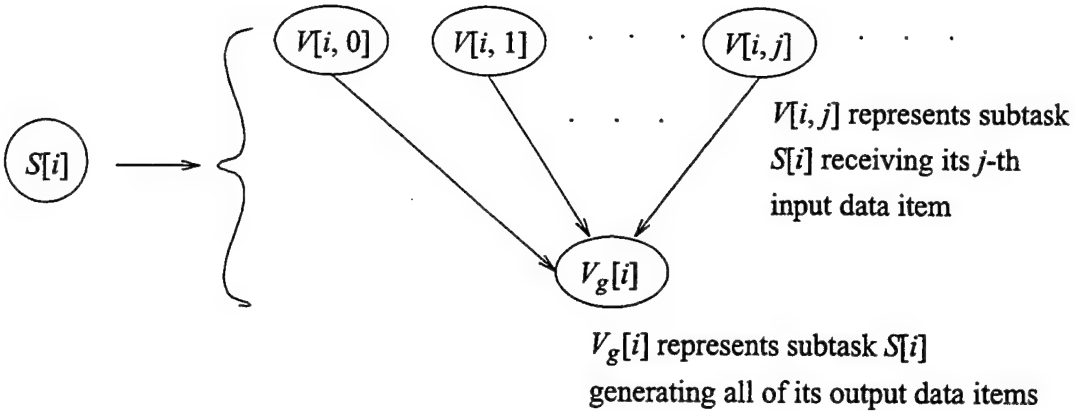

S[i]
V[i, 0]
V[i, 1]
V[i, j]
Vg[i]
V[i, j] represents subtask S[i] receiving its j-th input data item
Vg[i] represents subtask S[i] generating all of its output data items

Figure 2: The generation of the input and output data vertices and activate edges for $S[i]$.

one of the initial locations where d_k is stored before program execution, then add an edge from the *Source* vertex to $V[i_1, j_1]$ whose weight is the communication time interval needed to transfer that initial data element from the initial location $M[k_2]$ where it is stored to the machine assigned to $S[i_1]$.

Case B: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from the subtask that generates $Id[i_1, j_1]$.

If $0 \le i_2 < n$ and there is j_2, such that $Id[i_1, j_1] = Gd[i_2, j_2]$, then a directed edge with weight $D[k_2, Af(i_1), Id[i_1, j_1]]$ is added from $V_g[i_2]$ to $V[i_1, j_1]$. That is, if subtask $S[i_1]$'s j_1-th input data item $Id[i_1, j_1]$ is subtask $S[i_2]$'s j_2-th output data item $Gd[i_2, j_2]$, then add an edge from $V_g[i_2]$ to $V[i_1, j_1]$ whose weight is the communication time interval needed to transfer that data item from $M[k_2]$ to the machine assigned to $S[i_1]$.

Case C: $S[i_1]$ obtains its required input-data item $Id[i_1, j_1]$ by copying it from one of the other subtasks that have obtained that input-data item already.

If $0 \le i_2 < n$, and there is a j_2, such that $Id[i_1, j_1] = Id[i_2, j_2]$, then a directed edge with weight $D[k_2, Af(i_1), Id[i_1, j_1]]$ is added from $V[i_2, j_2]$ to $V[i_1, j_1]$. That is, if subtask $S[i_1]$'s j_1-th input data item $Id[i_1, j_1]$ is obtained by copying subtask $S[i_2]$'s j_2-th input data item $Id[i_2, j_2]$, then add an edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ whose weight is the communication time interval needed to transfer that data item from $M[k_2]$ to the machine assigned to $S[i_1]$.

For any input data vertex $V[i_1, j_1]$ $(0 \leq i_1 < n$ and $0 \leq j_1 < NI[i_1])$ for a given *DS*, one and only one case of A, B, or C can occur. Thus, any vertex $V[i_1, j_1]$ has one and only one parent vertex, which is specified by the given *DS*. Also, the weight of the edge between $V[i_1, j_1]$ and its unique parent vertex is the communication time interval needed for $S[i_1]$ to obtain $Id[i_1, j_1]$ from its source with respect to the given *Af*, *Sf*, and *DS*.

Step 4: For every $0 \leq i < n$, a directed edge with weight zero is added from $V[i, j]$ to $V_g[i]$ for all j, $0 \leq j < NI[i]$ (as shown in Figure 2). All the edges generated in this step are called activate edges.

As an example, suppose that for the specific application program *P* illustrated by the subtask flow graph shown in Figure 1, Table 1 lists its corresponding parameters. The initial data elements of *P* are d_0 and d_1; The generated data items of *P* are X_0, X_1, Y, Z_0, and Z_1. Note that initial data elements are named with lower case letters and generated data items with upper case letters. The result of applying the set of data-source functions defined by the subtask flow graph in Figure 1 is shown by Figure 3 (recall that is just one possible set of data-source functions).

If the *Gr* generated above is an acyclic graph, then the corresponding *DS* is defined as a valid set of data–source functions for the application program *P*. If the graph had a cycle, then deadlock would arise in the application program *P*, which makes *P* unschedulable. Readers should notice that the weight of each edge or vertex depends on *Af*, *Sf*, and *DS*. The validity of a particular *DS* is based on the subtask flow graph and is independent of the underlying *Af* and *Sf* for generating the specific *Gr*. For the rest of this paper, only valid sets of data-source functions will be considered.

Step 5: For each i_1 and i_2 $(0 \leq i_1 < n$ and $0 \leq i_2 < n)$, if $Af(i_1) = Af(i_2)$ and $Sf(i_1) = Sf(i_2) - 1$ (i.e., $S[i_1]$ and $S[i_2]$ are assigned to the same machine and $S[i_1]$ is executed immediately before

subtask	no. inputs	input data items	no. outputs	output data items
$S\,[0]$	$NI[0] = 1$	$Id\,[0, 0] = (-1, d_0)$	$NG[0] = 2$	$Gd\,[0, 0] = (0, X_0)$ $Gd\,[0, 1] = (0, X_1)$
$S\,[1]$	$NI[1] = 2$	$Id\,[1, 0] = (-1, d_0)$ $Id\,[1, 1] = (0, X_0)$	$NG[1] = 1$	$Gd\,[1, 0] = (1, Y)$
$S\,[2]$	$NI[2] = 2$	$Id\,[2, 0] = (0, X_0)$ $Id\,[2, 1] = (-1, d_1)$	$NG[2] = 2$	$Gd\,[2, 0] = (2, Z_0)$ $Gd\,[2, 1] = (2, Z_1)$
$S\,[3]$	$NI[3] = 3$	$Id\,[3, 0] = (-1, d_1)$ $Id\,[3, 1] = (1, Y)$ $Id\,[3, 2] = (2, Z_0)$	$NG[3] = 0$	
$S\,[4]$	$NI[4] = 2$	$Id\,[4, 0] = (0, X_1)$ $Id\,[4, 1] = (2, Z_1)$	$NG[4] = 0$	
$S\,[5]$	$NI[5] = 2$	$Id\,[5, 0] = (0, X_1)$ $Id\,[5, 1] = (2, Z_0)$	$NG[5] = 0$	

Table 1: Parameters for the subtask flow graph shown in Figure 1.

$S[i_2]$), a directed edge with weight zero is added from $V_g[i_1]$ to $V_g[i_2]$. The extended graph based on *Gr* and *Sf* after this step is defined as the execution graph *Ex* of *P*. For the example in Figure 1, one possible assignment function *Af* is: $Af(0) = 1$, $Af(1) = 2$, $Af(2) = 2$, $Af(3) = 1$, $Af(4) = 3$, and $Af(5) = 0$. One possible scheduling function *Sf* for this example and *Af* is: $Sf(0) = 0$, $Sf(1) = 0$, $Sf(2) = 1$, $Sf(3) = 1$, $Sf(4) = 0$, $Sf(5) = 0$, then the corresponding directed edges added by this step are shown by the dashed lines from $V_g[0]$ to $V_g[3]$ and from $V_g[1]$ to $V_g[2]$ in Figure 3. If the generated execution graph *Ex* is acyclic, then the corresponding scheduling function generates an operational program and is defined as a valid scheduling function. For the rest of this paper, only valid scheduling functions will be considered.

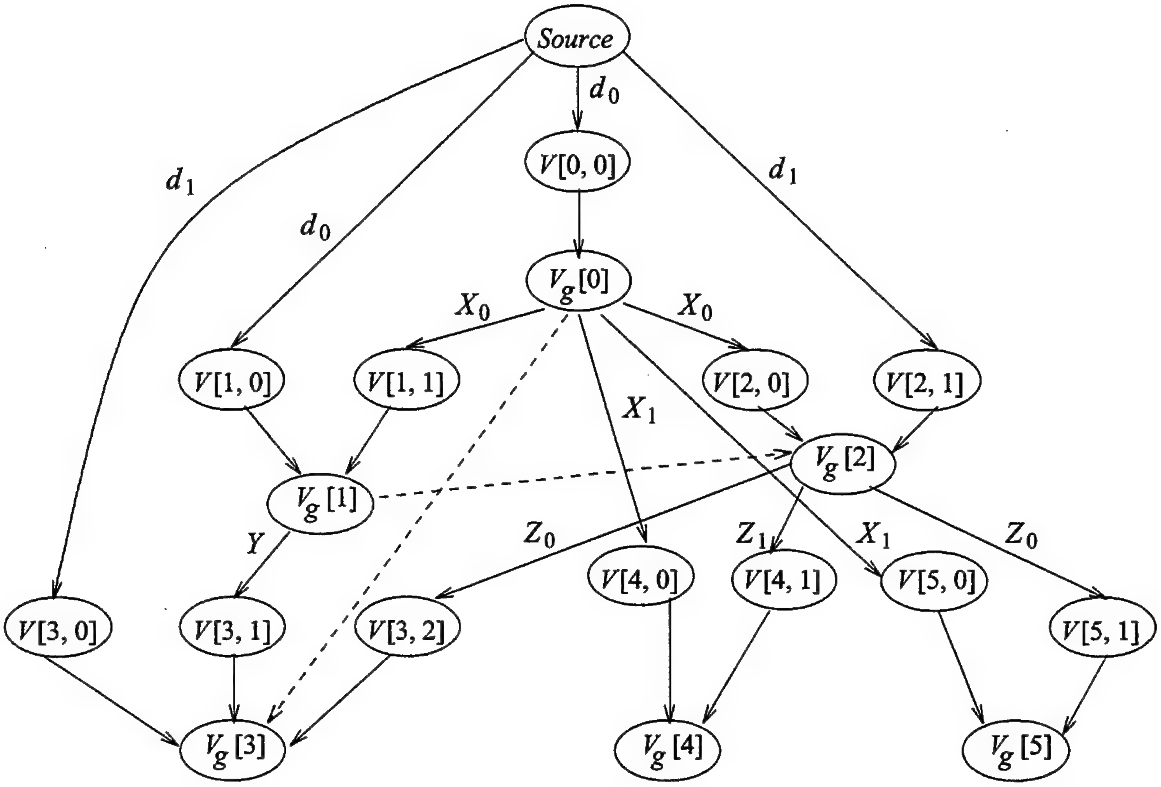


Figure 3: Generating an execution graph with respect to the given matching, scheduling, and data relocation schemes associated with the subtask flow graph shown in Figure 1.

Step 6: Each vertex $\underline{v}$ of *Ex* is associated with a $\underline{\text{starting time}}$ $\underline{ST(v)}$ and a $\underline{\text{finishing time}}$ $\underline{FT(v)}$ ($ST(v)$ and $FT(v)$ are random variables). From the definitions in Steps 4 and 5, the execution graph *Ex* generated is acyclic. Thus, there exists a topological sort [CoL90] of the vertices in *V*. Set $ST(Source) = 0$. $\underline{W(v)}$ is the weight of v (recall that each $V[i, j]$ is associated with a weight zero and each $V_g[i]$ is associated with a weight $C[i, Af(i)]$). Suppose that $\underline{v_k}$ is one of the immediate predecessors of v, $\underline{W(v_k, v)}$ is the weight of the direct edge from v_k to v. Then $ST(v)$ and $FT(v)$ can be derived inductively one by one in the order specified by the topological sort according to the following formulae:

$$ST(v) = \max_{k} \ \{FT(v_k) + W(v_k, v)\} \tag{1}$$

$$FT(v) = ST(v) + W(v). \tag{2}$$

Step 7: The total execution time of the application program *P* associated with an assignment function *Af*, a valid scheduling function *Sf*, and a valid set of data-source functions *DS* is defined by the following formula:

$$\text{Execution\_time}_P(Af\,,\ Sf\,,\ DS\,) = \max_{v \in V}\{FT(v)\}. \tag{3}$$

Suppose that $\underline{\text{E}\{x\}}$ denotes the expected value of a random variable *x*. The objective of matching, scheduling, and data relocation for HC is to find an assignment function Af^*, a valid scheduling function Sf^*, and a valid set of data-source functions DS^*, such that

$\text{E}\{\text{Execution\_time}_P(Af^*, Sf^*, DS^*)\} =$

$$\min_{Af,Sf,DS} \text{E}\{\text{Execution\_time}_P(Af\,, Sf\,, DS\,)\}. \tag{4}$$

Thus, the minimization of the expected value of the total execution time of an application program is the global optimization criterion of the optimization problem for HC described in Section 1 with respect to the stochastic model defined in Section 3.

It is assumed in this mathematical model that, if there is no data dependency between two subtasks *S*[*i*] and *S*[*j*], and they are assigned to be executed on two different machines by the assignment function *Af*, then *S*[*i*] and *S*[*j*] can be executed concurrently. Furthermore, the inter-machine communication step for one subtask to obtain one of its input data items can be overlapped with (a) inter-machine communication step(s) to obtain its other input data item(s), (b) the inter-machine communication steps of other subtasks to obtain their input data items, and (c) the computation steps of other subtasks. The distribution of each random variable *D*[*s*, *r*, *e*] indicates any time delay resulting from network or machine I/O conflicts.

As stated in Section 3, it is extremely difficult to obtain the exact distribution of *D*[*s*, *r*, *e*]. The purpose of the above topological sort based procedure is not for calculating

Execution_time$_P$(*Af*, *Sf*, *DS*) in practice due to this difficulty. Rather it is to define the global optimization criterion theoretically for the optimization problem of HC. The theorem presented in Section 6 is based on this defined Execution_time$_P$(*Af*, *Sf*, *DS*) with a known *Af*, *Sf*, and *DS* and provides a practical local optimization criterion for future data relocation heuristics.

5: A Procedure for Enumerating the Valid Options in Choosing Data Relocation Schemes

5.1: Overview

In Subsection 5.2, a procedure for enumerating all the valid options in choosing the data relocation schemes with respect to an arbitrary matching is described for subtask flow graphs without inter-subtask conditional and looping constructs. With the presence of the inter-subtask conditional and looping constructs, the same procedure presented in Subsection 5.2 is extended in Subsection 5.3 to enumerate the valid options in choosing the data relocation schemes. The material presented in this section defines the search space for the optimization problem based on the stochastic model of HC mentioned in Section 1. This search space enumerates the possible combinations of *Af*, *Sf*, and *DS* with respect to a specific subtask flow graph (or a specific HC application). The number of valid combinations (i.e., the size of the search space) denotes the complexity of the optimization problem. This defined search space also helps future data relocation heuristic developers to know all the valid options in choosing a data relocation scheme.

5.2: Description for Subtask Flow Graphs without Inter-Subtask Conditional and Looping Constructs

A directed graph *Dg*[*Af*] corresponding to a specific assignment function *Af* can be generated by connecting the vertices in *V* as follows (recall that *V* is a set that contains all the vertices generated for any specific application program *P* according to Steps 1 and 2 described in Section 4). This directed graph (via vertex and edge connectivity) illustrates all possible sources from where a subtask could fetch its individual input data item:

Step 1: For every i_1, j_1, i_2, and j_2, where $0 \leq i_1 < n$, $0 \leq i_2 < n$, $0 \leq j_1 < NI[i_1]$, $0 \leq j_2 < NI[i_2]$, and $i_1 \neq i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, a directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ and a directed edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ are added.

Step 2: For every i_1, j_1, i_2, and j_2, where $0 \leq i_1 < n$, $0 \leq i_2 < n$, $0 \leq j_1 < NG[i_1]$, and $0 \leq j_2 < NI[i_2]$, such that $Gd[i_1, j_1] = Id[i_2, j_2] = e$, a directed edge from $V_g[i_1]$ to $V[i_2, j_2]$ is added.

After the above Steps 1 and 2, each generated data item $Gd[i_1, j_1]$ of P corresponds to a fully connected graph of the set of vertices $\underline{VG\ [i_1, j_1]} = \{V[i_2, j_2] \mid Gd[i_1, j_1] = Id[i_2, j_2], 0 \leq i_2 < n, 0 \leq j_2 < NI[i_2]\}$. This corresponds to the set of input data vertices that need the generated data item $Gd[i_1, j_1]$. Also, $V_g[i_1]$ is connected uni-directionally (i.e., $V_g[i_1]$ is the starting point of each directed edge) to all the vertices in $VG[i_1, j_1]$.

Step 3: For every i, j, and k, such that $Id[i, j] = (-1, d_k)$, where $0 \leq i < n$, $0 \leq j < NI[i]$, and $0 \leq k < Q$, a directed edge from the *Source* vertex to $V[i, j]$ is added.

After the above Step 3, each initial data item $(-1, d_k)$ $(0 \leq k < Q)$ of P corresponds to a fully connected graph of the set of vertices $\underline{VI[k]} = \{V[i, j] \mid Id[i, j] = (-1, d_k)\}$ (i.e., the input data vertices that need the initial data element d_k). There is also a directed edge from the *Source* vertex to each vertex in $VI[k]$. All the edges generated in the above Steps 1, 2, and 3 are called <u>fetch</u> <u>edges</u>.

Figure 4 illustrates components of $Dg[Af]$ for the example discussed in Section 4, based on the subtask flow graph shown in Figure 1. Recall that the parameters of the above example subtask flow graph are listed by Table 1. After applying above Steps 1, 2, and 3, the edges (both solid and dashed lines) of $Dg[Af]$ in Figure 4 are fetch edges corresponding to the initial data elements d_0 and the generated data items X_0 and Z_0.

A directed graph $Dg[Af]$ can be generated by knowing only P and Af. After generating $Dg[Af]$, any algorithm for enumerating the spanning trees of a directed graph [CoL90] can be applied to the subgraphs of $Dg[Af]$ for (1) the set of vertices $\{V_g[i]\} \cup VG[i, j]$ $(0 \leq i < n$ and $0 \leq j < NG[i])$ and (2) the set of vertices $\{Source\} \cup VI[k]$ $(0 \leq k < Q)$. The roots of all possible span-

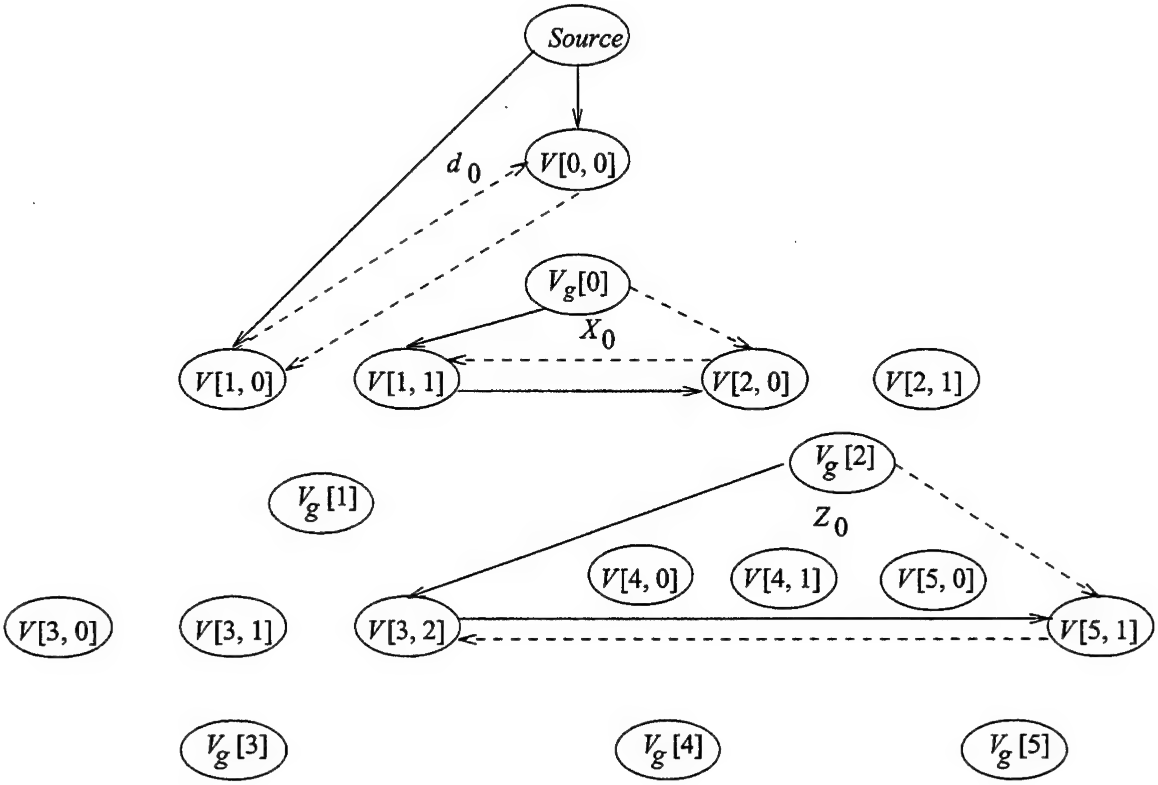


Figure 4: The d_0, X_0, and Z_0 components of *Dg*[*Af*], based on the subtask flow graph in Figure 1.

ning trees are $V_g[i]$ ($0 \leq i < n$) or the *Source* vertex, respectively. Each spanning tree corresponding to the set of vertices $\{V_g[i]\} \cup VG[i, j]$ specifies a valid data relocation scheme for the generated data item $Gd[i, j]$. Because the *Source* vertex can denote multiple locations where each initial data element d_k is stored before the execution of *P*, each spanning tree corresponding to the set of vertices $\{Source\} \cup VI[k]$ can specify a suite of valid data relocation schemes for the initial data element d_k. In the above generated spanning trees, if the parent vertex of $V[i_1, j_1]$ is $V[i_2, j_2]$ or $V_g[i_2]$, then $DS[i_1](j_1) = [i_2, Af(i_2)]$; and if the parent vertex of $V[i_1, j_1]$ is the *Source* vertex, then $DS[i_1](j_1) = [-1, q]$, where $M[q]$ is one of the initial locations of the corresponding initial data element. The solid lines in Figure 4 illustrate one spanning tree for each of d_0, X_0, and Z_0, respectively.

5.3: Description for Subtask Flow Graphs with Inter-Subtask Conditional and Looping Constructs

5.3.1: With the Presence of Inter-Subtask Conditional Constructs

In order to maintain a static analysis approach, it is assumed that the branching probabilities $\underline{P_{\text{then}}}$ and $\underline{P_{\text{else}}}$ for the "then" and "else" clauses of the input-data-dependent conditional constructs in the subtask flow graph are known and $P_{\text{then}} + P_{\text{else}} = 1$. Estimates of these two probabilities can be determined from empirical information or be supplied by the application users (such assumptions are typical in the literature, e.g., [Tow86]). Figure 5(a) shows an example in which there is an input-data-dependent conditional construct after $S[1]$. It is assumed that the left branch after $S[1]$ is the "then" clause and the right branch after $S[1]$ is the "else" clause of the corresponding input-data-dependent conditional construct.

The expected time for computing subtask $S[i]$ in the "then" clause of an input-data-dependent conditional construct on $M[Af(i)]$ is $P_{\text{then}} \cdot C[i, Af(i)]$. The expected time for computing subtask $S[i]$ in the "else" clause of an input-data-dependent conditional construct on $M[Af(i)]$ is $P_{\text{else}} \cdot C[i, Af(i)]$. Similarly, the inter-machine data transfer times for transferring subtasks' input and output data items inside an input-data-dependent conditional construct should be multiplied by their corresponding branching probability. For example, as shown in Figure 5(a), $(1, D_1)$ of $S[2]$ and $(4, D_5)$ of $S[5]$ are inside the input-data-dependent conditional construct, but $(0, D_0)$ of $S[1]$ and $(5, D_6)$ of $S[6]$ are not. With the above changes of the timing information, the topological sort based procedure presented in Section 4 can be used to determine the total execution time of a subtask flow graph with input-data-dependent conditional constructs.

With the presence of input-data-dependent conditional constructs in the subtask flow graph, the post-conditional locations of the input data items and output data items of the subtasks inside the "then" and "else" clauses cannot be determined at compile time (i.e., their locations will

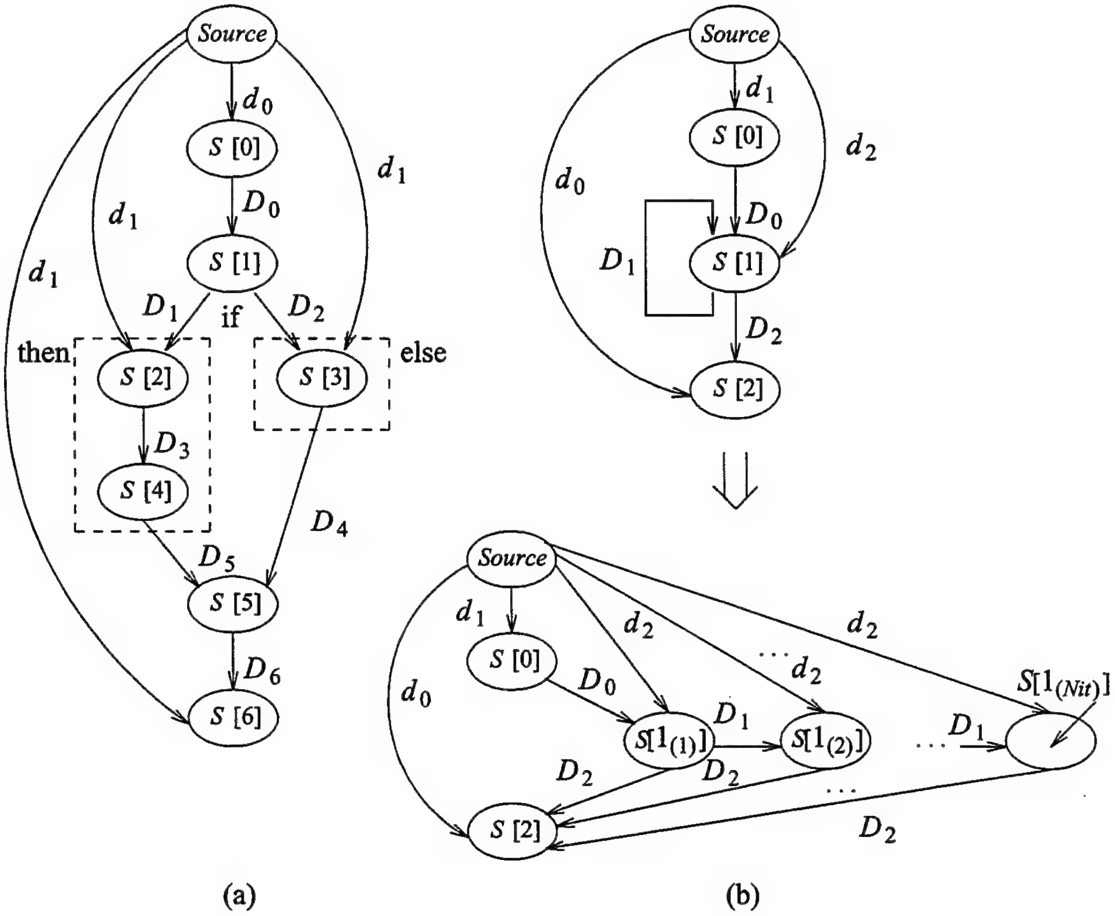


Figure 5: (a) Input-data-dependent conditional and (b) looping constructs in the subtask flow graphs (d_0, d_1, and d_2 are initial data elements, D_0 to D_6 are generated data items).

depend on the value of the conditional and how the clauses are executed at run time). The procedures for adding fetch edges to generate *Dg*[*Af*] presented in Subsection 5.2 must be modified to reflect the properties of input-data-dependent conditional constructs. For the ease of presentation, the following procedures are presented for the case of having only one input-data-dependent conditional construct in the subtask flow graph. For the case of having nested input-data-dependent conditional constructs and/or more than one input-data-dependent conditional construct in the same scope level of the application program, the same procedures can be extended inductively and applied due to the modular structure [BoM76] of the subtask flow graph.

Step 1: For each input or output data item d, an associated scope level $Scope[d]$ is defined. Subtasks' input and output data items that are not part of the input-data-dependent conditional construct all have their corresponding scope levels as "Outside." For example, $(-1, d_0)$ of $S[0]$ and $(5, D_6)$ of $S[6]$ in Figure 5(a) belong to this category. Alternatively, subtasks' input and output data items that are part of the input-data-dependent conditional construct all have their corresponding scope levels as "Inside." For example, $(1, D_1)$ of $S[2]$, $(-1, d_1)$ of $S[3]$, and $(2, D_3)$ of $S[4]$ in Figure 5(a) all belong to this category.

Step 2: Each input and output data item d is associated with a clause identifier $Cid[d]$. Subtasks' input and output data items that are not part of the input-data-dependent conditional construct all have their corresponding clause identifier as "Global." For example, both $(-1, d_0)$ of $S[0]$ and $(5, D_6)$ of $S[6]$ belong to this category. Subtasks' input and output data items that are part of the "then" clause of the input-data-dependent conditional construct have their corresponding clause identifier as "Then." For example, $(1, D_1)$ of $S[2]$, $(-1, d_1)$ of $S[2]$, and $(4, D_5)$ of $S[5]$ all belong to this category. Subtasks' input and output data items that are part of the "else" clause of the input-data-dependent conditional construct have their corresponding clause identifier as "Else." For example, $(1, D_2)$ of $S[3]$, $(-1, d_1)$ of of $S[3]$, and $(3, D_4)$ of $S[5]$ belong to this category.

Step 3: One extension of the definition of the scope level of a data item is described as follows. If for two data items $Id[i_1, j_1]$ and $Id[i_2, j_2]$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, $Scope[Id[i_1, j_1]] = Scope[Id[i_2, j_2]] =$ "Inside," $Cid[Id[i_1, j_1]] =$ "Then," $Cid[Id[i_2, j_2]] =$ "Else," and $Af(i_1) = Af(i_2) = i$, then reset $Scope[Id[i_1, j_1]] = Scope[Id[i_2, j_2]] =$ "Outside" and $Cid[Id[i_1, j_1]] = Cid[Id[i_2, j_2]] =$ "Global." The reason is that, because no matter what is the exact execution trace of the input-data-dependent conditional construct during run time, data item e will be available on machine $M[i]$ either via $S[i_1]$'s copy inside the "then" clause or $S[i_2]$'s copy inside the "else" clause. For example, if $Af(2) = Af(3) = i$, then $(-1, d_1)$ on machine $M[i]$ belongs to this category.

After the above three labeling steps for each data item of the subtasks in the application program, each data item d is associated with a scope level *Scope*[d] and a clause identifier *Cid*[d]. The following Step 4 is defined based on the above two augmented parameters of d.

Step 4: Step 1 in Subsection 5.2 should be modified as the following. Steps 2 and 3 in Subsection 5.2 are unchanged.

Case A: For every i_1, j_1, i_2, and j_2, where $0 \le i_1 < n$, $0 \le i_2 < n$, $0 \le j_1 < NI[i_1]$, $0 \le j_2 < NI[i_2]$, and $i_1 \ne i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, $Scope[Id[i_1, j_1]] = Scope[Id[i_2, j_2]]$, and $Cid[Id[i_1, j_1]] = Cid[Id[i_2, j_2]]$, a directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ and a directed edge from $V[i_2, j_2]$ to $V[i_1, j_1]$ are added.

Case B: For every i_1, j_1, i_2, and j_2, where $0 \le i_1 < n$, $0 \le i_2 < n$, $0 \le j_1 < NI[i_1]$, $0 \le j_2 < NI[i_2]$, and $i_1 \ne i_2$, such that $Id[i_1, j_1] = Id[i_2, j_2] = e$, $Scope[Id[i_1, j_1]] =$ "Outside," and $Scope[Id[i_2, j_2]] =$ "Inside", only one directed edge from $V[i_1, j_1]$ to $V[i_2, j_2]$ is added.

With the above four augmenting steps (compared with Steps 1, 2, and 3 in Subsection 5.2) for generating *Dg*[*Af*], subtask flow graphs with input-data-dependent conditional constructs can be handled properly. Those augmenting steps with scope levels and clause identifiers, enumerate all the possible sources for fetching each particular data item with the presence of the input-data-dependent conditional constructs in the subtask flow graph.

5.3.2: With the Presence of Inter-Subtask Looping Constructs

Similar to the case of having input-data-dependent conditional constructs, for the ease of presentation, the following procedures are presented for the case of having only one looping construct in the entire subtask flow graph. For the case of having nested looping constructs and/or more than one looping construct, the same procedures can be extended inductively and applied just as well due to the modular structure of the subtask flow graph, as discussed for the data conditional cases.

Suppose *Nit* is the maximum number of iterations that the looping construct will execute. $S[i_{(j)}]$ is the subtask that represents the number j iteration of subtask $S[i]$ that is inside a looping construct, for $1 \le j \le Nit$. It is assumed that the distribution for the number of iterations the looping construct will execute is known. Let $Lp[k]$ $(0 \le \ k \le Nit)$ is the probability that the looping construct will execute a total of k iterations, where $\sum_{k=0}^{Nit} Lp[k] = 1$. Then, $\tilde{Lp}[j] = \sum_{k=j}^{Nit} Lp[k]$ is the probability that iteration number j of the looping construct will be executed.

As shown in Figure 5(b), there is an input-data-dependent looping construct (containing $S[1]$) between $S[0]$ and $S[2]$. The initial cyclic subtask flow graph (due to the presence of the looping construct) is transformed into an acyclic one. A total of *Nit* copies of $S[1]$ are generated.

It is assumed that a matching scheme is given for all the subtasks (including $S[i_{(j)}]$'s) in the acyclic subtask flow graph generated by the above transformation. The time for computing iteration number j of $S[i]$ (i.e., $S[i_{(j)}]$) on machine $M[Af(i_{(j)})]$ is $\tilde{Lp}[j] \cdot C[i_{(j)}, Af(i_{(j)})]$. Similarly, the inter-machine data transfer time for transferring subtasks' input and output data items inside iteration number j of the looping construct should be multiplied by $\tilde{Lp}[j]$ as well. With the above changes of the timing information, the topological sort based procedure presented in Section 4 can be used to determine the total execution time of a subtask flow graph with looping constructs.

For the case of having a looping construct in the subtask flow graph, Step 1 in Subsection 5.2 must have the following two rules added. Steps 2 and 3 in Subsection 5.2 are unchanged.

(1) Subtask $S[i_{(j)}]$ (i.e., the iteration number j of $S[i]$ in the looping construct) can obtain its input data items by copying them from other subtasks that are in the iteration number k of the looping construct, where $k \le j$. $S[i_{(j)}]$ also can obtain its input data items by copying them from other subtasks that are not in the looping construct or from the *Source*.

(2) Subtask $S[k]$ that is not in the looping construct can only obtain its input data items by copying them from other subtasks that are not in the looping construct, from $S[i_{(j)}]$ where $\tilde{Lp}[j] = 1$, or from the *Source*.

6: A Greedy Approach to Establishing a Local Optimization Criterion for Developing Data Relocation Heuristics

In this section, a greedy algorithm based approach to establishing a local optimization criterion for developing data relocation heuristics is presented. This greedy strategy is established based on the mathematical model, the global optimization criterion, and search space described in Sections 3, 4, and 5, respectively, for the optimization problem in HC. The goal of this section is to show that an approach based on a greedy algorithm can establish a reasonable local optimization criterion for developing data relocation heuristics. Choosing *Af*, *Sf*, and *DS* to minimize the expected value of the total execution time based on a stochastic HC model is a complex optimization problem. However, developing heuristics to find suboptimal *Af*, *Sf*, and *DS* is necessary to use HC systems efficiently.

This section concentrates on developing data relocation heuristics to choose a suboptimal data relocation scheme for a given matching and scheduling to decrease the expected value of the total execution time of an HC application program. One of the techniques for developing heuristics is to achieve a reasonable level of global optimization using a well-evaluated local optimization criterion. The local optimization criterion for developing data relocation heuristics presented in this section is the minimization of the expected time when each subtask can start its computation after obtaining all of its input data items.

A greedy algorithm based approach for data relocation to achieving the above local optimization is to minimize the expected receiving time for each input data item of each subtask. Suppose that $\underline{rt(V[i,j])}$ is the random variable that specifies the receiving time of input data item $Id[i, j]$ for subtask $S[i]$. Then, the time when $S[i]$ can start its computation step after obtaining all of its input data items is $\max_{0 \le j < NI[i]} [rt(V[i, j])]$. The above greedy algorithm based approach states that, the minimization of $\mathrm{E}\{\max_{0 \le j < NI[i]} [rt(V[i, j])]\}$ can be achieved by the minimization of $\mathrm{E}\{rt(V[i,j])\}$ for all j, $0 \le j < NI[i]$. For an arbitrary set of random variables, this greedy strategy in general will *not* lead to the stated local optimization. That is, for a general set of random variables $G[j]$, $0 \leq j < J$, it is *not* always the case that the minimization of $E\{\max_{0 \leq j < J} G[j]\}$ can be achieved by the minimization of $E\{G[j]\}$ for all j, $0 \leq j < J$. But with the assumptions of the distributions of $rt(V[i,j])$ and $\underline{rt'(V[i,j])}$ corresponding to two different data relocation schemes DS and DS' shown in the next paragraph, this greedy approach can be proven to achieve the above stated local optimization.

The following assumptions about $rt(V[i,j])$ and $rt'(V[i,j])$ are made.

(1) $rt(V[i,j]) + k$ and $rt'(V[i,j]) + k'$ for a fixed i and j (where k and k' are arbitrary constants) belong to the same two-parameter family of random variables [CaB90]. Most of the common families of distributions for random variables, such as normal distribution, Gamma distribution, and Beta distribution, have this property.

(2) The variance of $rt(V[i,j])$ is equal to the variance of $rt'(V[i,j])$ for fixed i and j.

(3) For any data relocation scheme DS, $rt(V[i, j_1]) + c_1$ is independent of $rt(V[i, j_2]) + c_2$ ($j_1 \neq j_2$ and c_1 and c_2 are arbitrary constants).

Assumptions (1), (2), and (3) are all related to the statistical properties of $rt(V[i,j])$ and $rt'(V[i,j])$. If these assumptions are approximately satisfied in reality, the theorem that follows based on these assumptions is of practical as well as theoretical significance. For assumptions (1) and (2), because $rt(V[i,j])$ and $rt'(V[i,j])$ are two random variables for specifying the receiving times of the *same* data item (i.e., $Id[i,j]$) for $S[i]$, for the same Af and Sf, but corresponding to two different data relocation schemes, it is quite reasonable to assume that they have certain similar statistical properties (e.g., their variances, their families of distribution). For assumption (3), although $rt(V[i, j_1])$ and $rt(V[i, j_2])$ are defined for two different data items, if the inter-machine data transfer steps for $Id[i, j_1]$ and $Id[i, j_2]$ will impact each other or those two data items are generated by the same subtask, their corresponding receiving times by $S[i]$ can be correlated to each other. However, conditions exist under which the random variables can be treated as being independent of each other despite this type of correlation. The Kleinrock independence approximation for a data network in which there are many interacting transmission queues [Kle64] is a well-known method for describing this situation. This Kleinrock independence approximation is used here as the basis for assuming independence between $rt(V[i, j]) + c_1$ and $rt'(V[i, j]) + c_2$ that may technically be correlated. In [LiA97], similar assumptions are made about the execution time distributions for the individual subtasks for statically estimating the execution time distribution for an entire HC application program.

Theorem: For two different data relocation schemes DS and DS', with the same Af and Sf, and a fixed i $(0 \le i < n)$, suppose $\underline{X_j} = rt(V[i, j])$, $\underline{Y_j} = rt'(V[i, j])$, $\underline{X} = \max_j[X_j]$, and $\underline{Y} = \max_j[Y_j]$ $(0 \le j < NI[i])$, where X and Y are random variables for specifying the times when $S[i]$ receives all of its input data items with respect to DS and DS'. If $\mathrm{E}\{X_j\} \le \mathrm{E}\{Y_j\}$ for $0 \le j < NI[i]$, then $\mathrm{E}\{X\} \le \mathrm{E}\{Y\}$.

<u>Proof</u>: Suppose that the distribution function of a random variable w is F_w. Because $\mathrm{E}\{X_j\} \le \mathrm{E}\{Y_j\}$ for all j, there exists $c_j \ge 0$, such that $\mathrm{E}\{X_j\} + c_j = \mathrm{E}\{X_j + c_j\} = \mathrm{E}\{Y_j\}$. Due to assumption (2), $\mathrm{Var}\{X_j + c_j\} = \mathrm{Var}\{X_j\} = \mathrm{Var}\{Y_j\}$. Then, because of assumption (1), $F_{X_j + c_j} = F_{Y_j}$. From assumption (3), for $0 \le j < NI[i]$, $\{X_j + c_j\}$ and $\{Y_j\}$ are two sets of independent random variables. With the properties associated with the "max" operator over multiple independent random variables [CaB90], it can be shown that

$$F_{\max\{X_j + c_j\}} = \prod_{j=0}^{NI[i]-1} F_{X_j + c_j} = \prod_{j=0}^{NI[i]-1} F_{Y_j} = F_{\max\{Y_j\}}.$$

Therefore, $\mathrm{E}\{Y\} = \mathrm{E}\{\max_j[Y_j]\} = \mathrm{E}\{\max_j[X_j + c_j]\}$. Because $c_j \ge 0$, $\mathrm{E}\{Y\} = \mathrm{E}\{\max_j[X_j + c_j]\} \ge \mathrm{E}\{\max_j[X_j]\} = \mathrm{E}\{X\}$. Thus, $\mathrm{E}\{X\} \le \mathrm{E}\{Y\}$. □

Based on the above theorem, the greedy algorithm based approach that finds a data relocation scheme to minimize $\mathrm{E}\{rt(V[i, j])\}$ for $S[i]$ to obtain $Id[i, j]$ with respect to the same Af and Sf for all j $(0 \le j < NI[i])$ can also minimize the expected time when $S[i]$ receives all of its input data items (i.e., $\mathrm{E}\{X\}$) and is ready for its computation. The exact starting time and the cost of the computation for $S[i]$ (i.e., $ST(V_g[i])$ and $C[i, Af(i)]$) depend on the choice of Af and Sf.

The significance of the above theorem is that it shows a greedy algorithm based approach can establish a reasonable local optimization criterion for developing data relocation heuristics. Based on the above conclusion, in order to minimize the expected total execution time of an application program executed in a dedicated HC system (the global optimization criterion), data relocation heuristics should select the source for each input data item of $S[i]$, among all the valid options described in Section 5, such that its receiving time by $S[i]$ is as small as possible. With this greedy approach, the expected time when $S[i]$ can start its computation after obtaining all of its input data items (the local optimization criterion) can be minimized.

Given the approximation assumptions made, theoretically there exists a *DS* that can satisfy $\mathrm{E}\{X_j\} \leq \mathrm{E}\{Y_j\}$ for $0 \leq j < NI[i]$. However, in a real HC system, the inter-machine communication steps specified by the selected data relocation scheme for one subtask may impact the expected receiving time of input data items for other subtasks. Thus, the data relocation scheme that minimizes $E\{rt(V[i, j])\}$ for every $S[i]$ and all j ($0 \leq j < NI[i]$) may be hard to find or may not exist in practice. Trade-offs must be made to choose a suboptimal data relocation scheme, such that more input data items can be obtained by more subtasks as quickly as possible.

Because the $rt(V[i, j])$'s corresponding to different data relocation schemes are random variables, in general, the distributions of the $rt(V[i, j])$'s are needed for choosing a proper *DS* for obtaining $Id[i, j]$ for $S[i]$. But based on the above proven theorem with its underlying assumptions, only the expected values of $rt(V[i, j])$'s are needed to make such a decision, because the minimization of the expected time when $S[i]$ can obtain $Id[i, j]$ for each $0 \leq j < NI[i]$ can lead to the minimization of the expected time when $S[i]$ obtains all of its input data items and may start its computation. In practice, $E\{rt(V[i, j])\}$ corresponding to a particular *DS* is much easier to estimate than the probability distribution function of $rt(V[i, j])$. For example, for the networks using the popular and ubiquitous TCP/IP protocol, the inter-machine communication time component of $E\{rt(V[i, j])\}$ can be estimated by the value of the expected round trip time (RTT) delay between any two machines in the network kept by the TCP protocol on routers [Com91].

The minimization of the expected time when a selected subtask $S[i]$ can start its computation is adopted by many other matching and scheduling heuristics as their local optimization criterion [ElL90, IvO95, SiL93, WaA96]. The greedy approach, validated by the above theorem, for achieving this local optimization can be used by those matching and scheduling heuristics to intelligently select a data relocation scheme for $S[i]$ based on a theoretical stochastic HC model. Thus, coupled with the selection criterion for specifying the order of achieving the above local optimization for subtasks in the original matching and scheduling heuristic (e.g., priority-based for list scheduling), the greedy strategy for developing a data relocation heuristic presented in this paper can be expected to further decrease the inter-machine communication overhead of the given HC application program. For the matching and scheduling related heuristics that do not adopt the above local optimization criterion to achieve global optimization, choosing a data relocation scheme to minimize the expected time when a selected $S[i]$ can start its computation (realized by the presented greedy strategy) is still a reasonable approach to decrease the inter-machine communication overhead of an HC application. For example, in [WaS96], a genetic-algorithm-based heuristic for matching and scheduling applies the greedy strategy presented in this paper for selecting data relocation for a given matching and scheduling.

7: Summary

In an HC system, the subtasks of an application program P must be assigned to a suite of heterogeneous machines (the matching problem) and ordered (the scheduling problem) to utilize computational resources effectively. The matching and scheduling solutions presented in the literature, in general, concentrate on decreasing the computation time of P. The inter-machine communication time of P is impacted by the scheme for distributing the initial data elements and the generated data items of P to different subtasks (the data relocation problem).

The inter-machine communication time in an HC system can have a significant impact on overall system performance, so that any technique that can be used to reduce this time is important. This paper focused on the data relocation scheme to decrease the inter-machine communication time for given matching and scheduling schemes, when the possible concurrent execution of multiple subtasks on different machines is considered.

This paper concentrates on theoretical aspects of data relocation using a stochastic HC model. The optimization problem for minimizing the total execution time of an application program executed in a dedicated HC system with respect to matching, scheduling, and data relocation is completely defined. This theoretical definition is based on the stochastic mathematical model, the global optimization criterion, and the search space described in Sections 3, 4, and 5, respectively. The cases in which the application programs may include inter-subtask conditional and looping constructs are considered. The practical application of the above theoretical results is demonstrated by the theorem shown in Section 6 that validates a greedy algorithm based approach can establish a reasonable local optimization criterion for developing data relocation heuristics. The validation is provided by a theoretical proof based on a set of common assumptions about the underlying HC system and application program. The stochastic HC model presented, the local optimization criterion established by the greedy approach, and the search space defined for choosing valid data relocation schemes can help developers of future data relocation heuristics.

Acknowledgments: The authors thank John K. Antonio, Gayathri Krishnamurthy, Mark B. Kulaczewski, Yan A. Li, and Janet M. Siegel for their valuable comments. A preliminary version of portions of this material was presented at the 6th Heterogeneous Computing Workshop. Shoukat Ali assisted in the preparation of the final manuscript.

Appendix: Glossary of Notation

Af	assignment function (assigns subtasks of application program P to machines)
$C[i, j]$	computation time of subtask i on machine j
$Cid[d]$	clause identifier for data item d
$Dg[Af]$	directed graph (corresponding to program P) showing data transfer options based on a given Af
d_k	k-th initial data element of the application program P
$DS[i](j)$	source of the j-th input data item for subtask i
$D[s, r, e]$	time for transferring data item e from machine s to machine r
Ex	generated execution graph corresponding to a particular Af, Sf, and DS
$FT(v)$	finishing time of data transfer step (associated with an input data vertex v) and computation step (associated with an output data vertex v)
$G[i]$	generated output data set of subtask i
$Gd[i, j]$	j-th generated output data item of subtask i
Gr	generated graph corresponding to a particular Af and DS
$I[i]$	input data set of subtask i
$Id[i, j]$	j-th input data item of subtask i
$Lp[k]$	probability that a looping construct will execute a total of k iterations
$\tilde{Lp}[j]$	probability that iteration number j of a looping construct will be executed
$M[j]$	j-th machine in the HC system, $0 \leq j < m$
m	number of machines in the HC system
n	number of subtasks in the application program P

$NG[i]$	number of output data items generated by subtask i
Nit	maximum number of iterations that a looping construct will execute
$NI[i]$	number of input data items required by subtask i
$NS[j]$	number of subtasks assigned to be executed on machine j
P_{then}	branching probability for the ''then'' clause of the inter-subtask input-data-dependent conditional construct
P_{else}	branching probability for the ''else'' clause of the inter-subtask input-data-dependent conditional construct
Q	number of initial data elements for the application program P
$rt(V[i, j])$	random variable that specifies the receiving time of input data item $Id[i, j]$ for subtask $S[i]$
$Scope[d]$	scope level of data item d
Sf	scheduling function associated with the application program P
$S[i]$	i-th subtask of an application program P, $0 \leq i < n$
$S[i_{(j)}]$	subtask that represents the number j iteration of subtask $S[i]$ that is inside a looping construct
$ST(v)$	starting time of data transfer step (associated with an input data vertex v) and computation step (associated with an output data vertex v)
$V_g[i]$	output data vertex of subtask i
$V[i, j]$	j-th input data vertex of subtask i
$VG[i, j]$	set of input data vertices that need the generated data item $Gd[i, j]$
$VI[k]$	set of input data vertices that need the initial data element d_k
$W(v)$	weight of an input or output data vertex v
$W(v_k, v)$	weight of the direct edge from v_k to v, where v_k is one of the immediate predecessors of v

References:

[BoM76] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Elsevier Science Publishing Co., Inc., New York, NY, 1976.

[CaB90] G. Casella and R. L. Berger, *Statistical Inference*, Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA, 1990.

[ChE93] S. Chen, M. M. Eshaghian, A. Khokhar, and M. E. Shaaban, ''A selection theory and methodology for heterogeneous supercomputing,'' *2nd Workshop on Heterogeneous Processing*, Apr. 1993, pp. 15-22.

[CiL95] M. Cierniak, W. Li, and M. J. Zaki, ''Loop scheduling for heterogeneity,'' *4th IEEE Int'l Symp. on High-Performance Distributed Computing*, Aug. 1995, pp. 78-85.

[CoL90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, MA, 1990.

[Com91] D. E. Comer, *Internetworking with TCP/IP, Volume I: Principles, Protocols, and Architecture*, Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1991.

[ElL90] H. El-Rewini and T. G. Lewis, ''Scheduling parallel program tasks onto arbitrary target machines,'' *J. of Parallel and Distributed Computing*, Vol. 9, No. 2, June 1990, pp. 138-153.

[Fre89] R. F. Freund, ''Optimal selection theory for superconcurrency,'' *Supercomputing '89*, Nov. 1989, pp. 699-703.

[FrG98] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, ''Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet,'' *7th Heterogeneous Computing Workshop,* Mar. 1998, pp. 184-199.

[FrS93] R. F. Freund and H. J. Siegel, ''Heterogeneous processing,'' *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[GhY93] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78-86.

[IvO95] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop*, Apr. 1995, pp. 93-100.

[KhP92] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous supercomputing: problems and issues," *1st Workshop on Heterogeneous Processing*, Mar. 1992, pp. 3-12.

[KhP93] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[Kle64] L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, NY, 1964.

[LiA97a] Y. A. Li and J. K. Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," *6th Heterogeneous Computing Workshop*, Apr. 1997, pp. 172-184.

[LiA97b] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *J. of Parallel and Distributed Computing*, Vol. 44, No. 1, July 1997, pp. 35-52.

[NaY94] B. Narahari, A. Youssef, and H. A. Choi, "Matching and scheduling in a generalized optimal selection theory," *3rd Heterogeneous Computing Workshop*, Apr. 1994, pp. 3-8.

[SiA96] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[SiD97] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software Support for Heterogeneous Computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[SiL93] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 2, Feb. 1993, pp. 75-87.

[TaS97] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 8, No. 8 Aug. 1997, pp. 857-871.

[Tow86] D. Towsley, "Allocating programs containing branches and loops within a multiple processor system," *IEEE Trans. on Software Engineering*, Vol. SE-12, No. 10, Oct. 1986, pp. 1018-1024.

[WaA96] D. W. Watson, J. K. Antonio, H. J. Siegel, R. Gupta, and M. J. Atallah, "Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment," *5th Heterogeneous Computing Workshop*, Apr. 1996, pp. 24-37.

[WaK92] M. Wang, S.-D. Kim, M. A. Nichols, R. F. Freund, H. J. Siegel, and W. G. Nation, "Augmenting the optimal selection theory for superconcurrency," *1st Workshop on Heterogeneous Processing*, Mar. 1992, pp. 13-22.

[WaS97] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. of Parallel and Distributed Computing,* Special Issue on Parallel Evolutionary Computing, Vol. 47, No. 1, Nov. 25, 1997, pp. 8-22.

[YaK94] J. Yang, A. Khokhar, S. Sheikh, and A. Ghafoor, "Estimating execution time for parallel tasks in heterogeneous processing (HP) environment," *3rd Heterogeneous*

Computing Workshop, Apr. 1994, pp. 23-28.

[ZhY95] X. Zhang and Y. Yan, "Modeling and characterizing parallel computing performance on heterogeneous networks of workstations," *7th IEEE Symp. on Parallel and Distributed Processing*, Oct. 1995, pp. 25-34.

1 Mapping of Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach

MITCHELL D. THEYS*, TRACY D. BRAUN[†], YU-KWONG KWOK[‡],
HOWARD JAY SIEGEL[†], and ANTHONY A. MACIEJEWSKI[†]

*University of Illinois at Chicago
Electrical Engineering and Computer Science Department (MC 154)
851 S. Morgan St. RM 1120
Chicago, IL 60607-7053, USA
mtheys@eecs.uic.edu

[†]Purdue University
School of Electrical and Computer Engineering
West Lafayette, IN 47907-1285, USA
{tdbraun,hj,maciejew}@purdue.edu

[‡]The University of Hong Kong
Department of Electrical and Electronic Engineering
Pokfulam Road, Hong Kong
ykwok@eee.hku.hk

ABSTRACT

In a mixed-machine, distributed, heterogeneous computing (HC) environment, there is a suite of high-performance machines with different computational capabilities. These machines are interconnected by high-speed links. Such a suite of machines can be used to execute a single application, whose

This research was supported by the DARPA/ITO Quorum Program through GSA subcontract number GS09K99BH0250, an AFCEA Fellowship, the Hong Kong Research Grants Council under contract numbers HKU7124/99E and HKUST6076/97E, and a research initiation grant from the HKU CRCG.

subtasks have diverse execution requirements, or to execute a meta-task, which is a collection of independent tasks with different computational needs. This chapter discusses three different genetic-algorithm-based approaches used for solving the HC mapping problem: matching subtasks or independent tasks to machines and scheduling their execution in an HC environment. The first approach uses a genetic algorithm in a static, off-line mode to map subtasks and schedule the data transfers among subtasks. The second approach is for using a set of such static mappings in a dynamic, on-line manner to adapt to changing scenarios. The third approach uses a genetic algorithm in a static, off-line mode to map the independent tasks of a meta-task, and compares the results with several other meta-task mapping techniques.

1.1 INTRODUCTION

Different portions of an application task often require different types of computation. In general, it is impossible for a single machine architecture with its associated compiler, operating system, and programming tools to satisfy all the computational requirements in such an application equally well. One type of heterogeneous computing (HC) environment consists of a suite of different types of machines, high-speed interconnections, interfaces, operating systems, communication protocols, and programming environments that provides a variety of architectural capabilities. Such an HC environment can be orchestrated to perform an application that has diverse execution requirements [17, 20, 22, 32, 46, 47, 50]. An application task can be decomposed into subtasks, where each subtask is computationally homogeneous (well suited to a single machine), and different subtasks may have different machine architectural requirements. These subtasks may have data dependencies among them. A group of independent tasks, known as a meta-task, can also be executed in the HC environment. The tasks in a meta-task have no data dependencies among them, and may have different architectural requirements.

Once the set of subtasks or independent tasks to be executed is known, the following decisions must be made: matching, i.e., assigning subtasks or independent tasks to machines, and scheduling, i.e., ordering subtask or independent task execution for each machine and (for subtasks) ordering intermachine data transfers. The resulting matching and scheduling scheme is called a mapping. In the studies reported here, the primary goal of HC is to find a mapping that will achieve the minimal completion time, i.e., the minimal overall execution time of the application task or meta-task on the machine suite.

It is well known that such a mapping problem is, in general, NP-complete [19, 29]. A number of approaches to different aspects of this problem have been proposed (e.g., [18, 20, 30, 38, 51, 55]). These approaches include both static and dynamic heuristics. Static heuristics are executed off-line, in a planning mode, when the application task or meta-task to map is known in advance. Dynamic heuristics are executed on-line, in real-time, and can make use of feedback from the HC system (e.g., [37]). Heuristics developed to perform the mapping function are often difficult to compare because of different underlying assumptions in the original studies of each heuristic (e.g., number and types of machines in the HC suite) [5]. Various researchers have considered the use of genetic-algorithm-based approaches for this mapping function (e.g., [45, 48, 52, 61]). This chapter summarizes three particular genetic-algorithm-based approaches, each for a different HC environment.

Two static approaches and one dynamic approach used for solving the mapping problems for different HC environments are discussed. The first approach decides the subtask to machine assignments, orders the execution of the subtasks assigned to each machine, and schedules the data transfers among subtasks [58]. The characteristics of this approach include: separation of the matching and the scheduling representations, independence of the chromosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. The computation and communication overlap is limited only by inter-subtask data dependencies and machine/network availability. This genetic-algorithm-based approach can be applied to performing the mapping in a variety of HC systems. It is applicable to the static mapping of production jobs and can be readily used to collectively schedule a set of tasks that are decomposed into subtasks.

The second approach focuses on a particular application domain (iterative automatic target recognition (ATR) tasks) and an associated specific class of dedicated heterogeneous parallel hardware platforms. For the computational environment considered, a methodology for real-time, on-line, input-data dependent remapping of the application subtasks to the processors (machines) in the heterogeneous parallel hardware platform using a previously stored off-line, statically determined, mapping is presented [8, 34, 35]. The operating system makes a heuristic-based decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system selects a previously determined and stored genetic-algorithm-based mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking).

The third approach examines the mapping of meta-tasks to machines in an HC suite. It is assumed in this approach that each machine executes a single task at a time, in the order in which the tasks arrive, and there are no dependencies among the tasks. Because of these assumptions, scheduling is simplified and the resulting solutions of the mapping heuristics focus more on finding an efficient matching of tasks to machines. Eleven different static mapping heuristics from the literature were implemented and compared by simulation studies under one set of common assumptions [6]. One of these heuristics was a genetic algorithm, providing insight into the relative performance of genetic algorithms to other mapping techniques.

The organization of this chapter is as follows. The details of the problem and assumptions made for each of the approaches are presented in Section 1.2. Section 1.3 briefly describes the basic steps of any genetic algorithm. In Section 1.4, a summary of the results from the static subtask mapping approach is presented [58]. Section 1.5 discusses results obtained from the on-line use of off-line derived mappings approach [8, 34]. Finally, Section 1.6 details experiences from implementing the static meta-task mapping approach [6].

The research reported in this chapter was supported in part by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks) [26]. MSHN is a collaborative research effort among the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested quality of service. The heuristics developed in this paper or their derivatives may be included in the MSHN prototype.

1.2 PROBLEM DESCRIPTIONS

1.2.1 Static Matching and Scheduling of Subtasks

There are many open research problems in the field of HC [36, 46]. To isolate and focus on the mapping problem, researchers typically make assumptions about other components of an overall HC system (e.g., [45, 48]). This subsection considers the assumptions made in [58] for the static mapping of subtasks.

It is assumed that the application task is written in some machine-independent language (e.g., [59]). It is also assumed that an application task is decomposed into multiple subtasks and the data dependencies among them are known and are represented by a directed acyclic graph (DAG). If intermachine data transfers are data dependent, then some set of expected data transfers must be assumed. The estimated expected execution time for each subtask on each machine is assumed to be known *a priori*. Finding the estimated expected execution times for subtasks is another research problem, which is beyond the scope of this paper. Approaches based on task profiling and analytical benchmarking are surveyed in [32, 46, 47]. The HC system is assumed to have operating system support for executing each subtask on the machine it is assigned and for performing inter-machine data transfers as scheduled by this genetic-algorithm-based approach.

In the type of HC system considered here, an application task is decomposed into a set of subtasks $\underline{S}$. Define $|S|$ to be the number of subtasks in the set, and s_i to be the i-th subtask, $0 \leq i < |S|$. An HC environment consists of a set of machines $\underline{M}$. Define $|M|$ to be the number of machines and m_j to be the j-th machine, $0 \leq j < |M|$. The global data items (gdis), i.e., data items

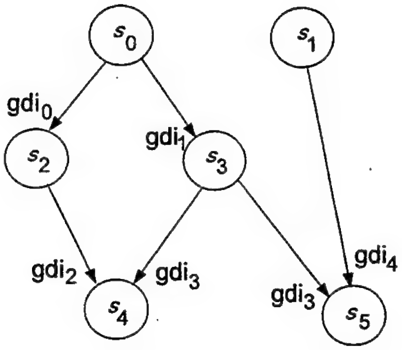


Fig. 1.1 An example SPDAG.

that need to be transferred between subtasks, form a set G. Define $|G|$ to be the number of items and gdi$_k$ to be the k-th global data item, $0 \leq k < |G|$.

It is assumed that for each global data item there is a single subtask that produces it (producer) and there are some subtasks that need this data item (consumers). Hence, the task is represented by a single-producer directed acyclic graph (SPDAG). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred over it. Figure 1.1 shows an example SPDAG.

The following further assumptions are made for the static subtask mapping problem. One is the exclusive use of the HC environment for the application; and that the genetic-algorithm-based mapper is in control of the HC machine suite. Another is non-preemptive subtask execution. Also, all input data items of a subtask must be received before its execution can begin, and none of its output data items are available until the execution of this subtask is finished. If a data conditional is based on input data, it is assumed to be contained inside a subtask. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a subtask. These restrictions help make the mapping problem more manageable and solving this problem under these assumptions is a significant step toward solving the general mapping problem.

For any static heuristic, it is assumed that an accurate estimate of the expected execution time for each subtask on each machine is known prior to execution and contained within an $|M| \times |S|$ ETC (expected time to compute) matrix. ETC(i, j) is the estimated execution time for subtask s_i on machine m_j. (These times are assumed to include the time to move the executables and initial data (not gdis) associated with subtask s_i to machine m_j when necessary.) The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems (e.g., [23, 31, 48]).

1.2.2 Semi-Static Matching and Scheduling of Subtasks

In [7, 8], a detailed design of an Intelligent Operating System (IOS) is proposed for a particular application domain in which: (1) an iterative application is to be mapped onto an associated specific type of dedicated heterogeneous parallel hardware platform; and (2) the execution of each iteration can be represented by a DAG of subtasks. To minimize the execution time of such an iterative application in a heterogeneous parallel computing environment, an appropriate mapping scheme is needed. However, when some of the characteristics of the application subtasks are unknown *a priori* and will change from iteration to iteration during execution-time, it may not be feasible or desirable to use the same off-line derived mapping throughout the whole execution of the application.

In such situations, a semi-static methodology [7, 8, 35] may be employed, that starts with an initial mapping but dynamically decides whether to remap the application with a mapping previously determined off-line. This is done by observing, from one iteration to another, the effects of the changing characteristics of the application's input data, called dynamic parameters, on the application's execution time. That is, the IOS will be able to make a heuristically-determined decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the IOS will be able to select a pre-computed and stored mapping that is appropriate for the given state of the application (e.g., the number of objects the ATR system is currently tracking). This remapping process will, in general, require a certain system reconfiguration time for relocating the data and program modules.

The semi-static method differs considerably from other real-time HC mapping techniques in that it involves the on-line, real-time use of off-line, precomputed mappings. This is significant because it is possible for off-line heuristics to have much longer execution times to search for a good solution than what is practical for an on-line heuristic. Thus, with the semi-static method, the mapping quality of an off-line, time-consuming heuristic can be approached at real-time speeds. As detailed in Subsection 1.5.6, the GA described in Section 1.4 is enhanced to be used as an off-line heuristic for determining high-quality mappings for on-line use. In an extensive simulation study, it was found that the semi-static method is much more effective than a purely dynamic approach.

1.2.3 Static Matching and Scheduling for Meta-Tasks

This subsection considers the assumptions made in [6] for the static mapping of meta-tasks. Recall from Section 1.1 that a meta-task is defined as a collection of independent tasks with no data dependencies (a given task, however, may have subtasks and dependencies among the subtasks). For this case study, it is assumed that static (i.e., off-line or predictive) mapping of meta-tasks is being performed. (In some systems, all tasks and subtasks in a meta-task, as defined above, are referred to as just tasks.)

It is also assumed that each machine executes a single task at a time, in the order in which the tasks arrived. Because there are no dependencies among the tasks, scheduling is simplified, and thus the resulting solutions of the mapping heuristics focus more on finding an efficient matching of tasks to machines. It is also assumed that the size of the meta-task T (number of tasks to execute) is $|T|$, and the number of machines in the HC environment is $|M|$, and both are static and known *a priori*. Expected task execution times are specified in an $|M| \times |T|$ ETC matrix, analogous to the one defined in Subsection 1.2.1 for subtasks. It is assumed that the HC system is dedicated for the meta-task, and controlled by the mapper.

1.3 GENETIC ALGORITHM OVERVIEW

Genetic algorithms (GAs) are a useful heuristic approach to finding near-optimal solutions in large search spaces [14, 25, 27]. There are a great variety of approaches to GAs; many are surveyed in [40, 49]. The following is a brief overview of GAs to provide background for the description of the proposed approaches.

The first step necessary to employ a GA is to encode some of the possible solutions to the optimization problem as a set of strings (chromosomes). Each chromosome represents one solution to the problem, and a set of chromosomes is referred to as a population. The next step is to derive an initial population. A random set of chromosomes is often used as the initial population. Some specified chromosomes can also be included as seeds. This initial population is the first generation from which the evolution begins.

The third step is to evaluate the quality of each chromosome. Each chromosome is associated with a fitness value, which in this case is the completion time of the solution (mapping) represented by this chromosome (i.e., the expected execution time of the application task or meta-task if the mapping specified by this chromosome were used). The objective of the GA search is to find a chromosome that has the optimal fitness value. The selection process is the next step. In this step, each chromosome is eliminated or duplicated (one or more times) based on its relative quality. The population size is typically kept constant.

Selection is followed by the crossover step. With some probability, pairs of chromosomes are selected from the current population and some of their corresponding components are exchanged to form two valid chromosomes, which may or may not already be in the current population. After crossover, each string in the population may be mutated with some probability. The mutation

```
GA_matching_scheduling {
   initial population generation;
   evaluation;
   while (stopping criteria not met) {
        selection;
        crossover;
        mutation;
        evaluation;
   }
   output the best solution found;
 }
```

Fig. 1.2 General procedure for a genetic algorithm, based on [49].

process transforms a chromosome into another valid chromosome that may or may not already be in the current population. The new population is then evaluated. If none of the stopping criteria has been met, the new population goes through another cycle (iteration) of selection, crossover, mutation, and evaluation. These cycles continue until one of the stopping criteria is met.

In summary, the following items must be determined to implement a GA for a given optimization problem: (1) an encoding, (2) an initial population, (3) an evaluation using a particular fitness function, (4) a selection mechanism, (5) a crossover mechanism, (6) a mutation mechanism, and (7) a set of stopping criteria. The outline of the GA-based approach is shown in Figure 1.2. Details of three GA-based approaches will be discussed in the following sections.

1.4 STATIC MATCHING AND SCHEDULING OF SUBTASKS

1.4.1 Introduction

This section discusses the GA-based approach found in [58] for the static mapping of subtasks. This section presents the details about the chromosome representation used, how population generation was performed, the mutation and crossover operators used, and comparisons with nonevolutionary approaches.

1.4.2 Chromosome Representation

In this GA-based approach, each chromosome consists of two parts: the matching string and the scheduling string. Let $\underline{\text{mat}}$ be the $\underline{\text{matching string}}$, which is a vector of length $|S|$, such that $\underline{\text{mat}(i)} = m_j$, where $0 \leq i < |S|$ and $0 \leq j < |M|$, i.e., subtask s_i is assigned to machine m_j.

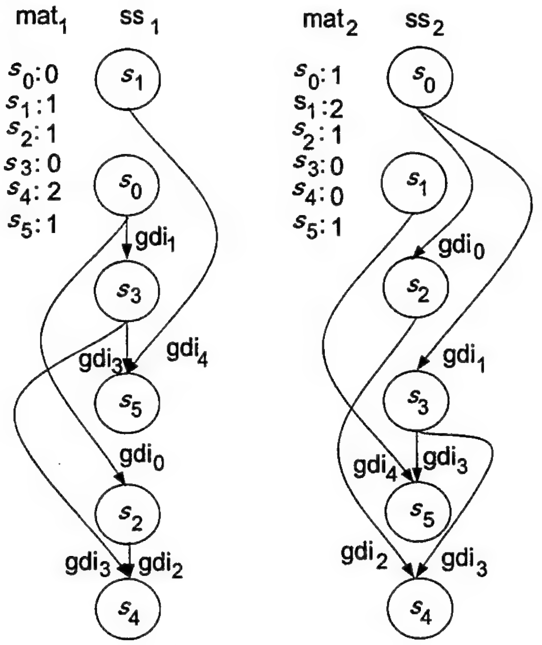


Fig. 1.3 Two chromosomes from the SPDAG in Figure 1.1.

The scheduling string is a topological sort [12] of the SPDAG, i.e., a total ordering of the nodes (subtasks) in the SPDAG that obeys the precedence constraints. Define ss to be the scheduling string, which is a vector of length $|S|$, such that ss(k)= s_i, where $0 \leq i, k < |S|$, and each s_i appears only once in the vector, i.e., subtask s_i is the k-th subtask in the scheduling string. Because it is a topological sort, if ss(k) is a consumer of a global data item produced by ss(j), then $j < k$. The scheduling string gives an ordering of the subtasks that is used by the evaluation step.

Then in this GA-based approach, a chromosome is represented by a two-tuple [mat,ss]. Thus, a chromosome represents the subtask-to-machine assignments (matching) and the execution ordering of the subtasks assigned to the same machine. The scheduling of the global data item transfers and the relative ordering of subtasks assigned to different machines are determined by the evaluation step. Figure 1.3 illustrates two different chromosomes for the SPDAG in Figure 1.1, for $|S| = 6$, $|M| = 3$, and $|G| = 5$.

1.4.3 Initial Population Generation

In the initial population generation step, a predefined number of chromosomes are generated, the collection of which forms the initial population. When generating a chromosome, a new matching string is obtained by randomly assigning each subtask to a machine. To form a scheduling string, the SPDAG is first topologically sorted to form a basis scheduling string. Then, for each chromosome in the initial population, this basis string is mutated a random number of times (between one and the number of subtasks) using the scheduling string mutation operator to generate the ss vector (which is a valid topological sort of the given SPDAG). Furthermore, it is common in GA applications to incorporate solutions from some non-evolutionary heuristics into the initial population, which may reduce the time needed for finding a satisfactory solution [14]. In this GA-based approach, along with those chromosomes representing randomly generated solutions, the initial population also includes a seed chromosome that represents the solution from a non-evolutionary baseline heuristic (see Subsection 1.4.8).

Each newly generated chromosome is checked against those previously generated. If a new chromosome is identical to any of the existing ones, it is discarded and the process of chromosome generation is repeated until a unique new chromosome is obtained.

1.4.4 Selection

In this step, the chromosomes in the population are first ordered (ranked) by their fitness values from the best to the worst. Those having the same fitness value are ranked arbitrarily among themselves. Then a rank-based roulette wheel selection scheme is used to implement the selection step [27, 49]. In the rank-based selection scheme, each chromosome is allocated a sector on a roulette wheel. Let P denote the population size and A_i denote the angle of the sector allocated to the i-th ranked chromosome. The 0-th ranked chromosome is the fittest and has the sector with the largest angle A_0; whereas the $(P-1)$-th ranked chromosome is the least fit and has the sector with the smallest angle A_{P-1}. The ratio of the sector angles between two adjacently ranked chromosomes is a constant $R = A_i/A_{i+1}$, where $0 \leq i < P-1$. If the 360 degrees of the wheel are normalized to one, then $A_i = R^{P-i-1} \times (1-R)/(1-R^P)$ where $R > 1$, $0 \leq i < P$, and $0 < A_i < 1$.

The selection step generates P random numbers, ranging from zero to one. Each number falls in a sector on the roulette wheel and a copy of the owner chromosome of this sector is included in the next generation. Because a better solution has a larger sector angle than that of a worse solution, there is a higher probability that one or more copies of this better solution will be included in the next generation. In this way, the population for the next generation is determined. Thus, the population size is always P, and it is possible to have multiple copies of the same chromosome.

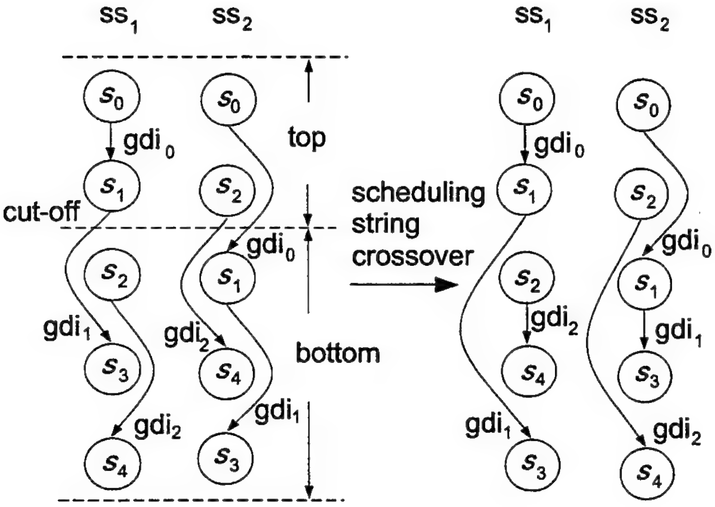


Fig. 1.4 A scheduling string crossover example.

This GA-based approach also incorporates elitism [41]. At the end of each iteration, the best chromosome is always compared with the previous best (elite) chromosome, a copy of which is stored separately from the population. If the best chromosome is better than the elite chromosome, a copy of it becomes the elite chromosome. If the best chromosome is not as good as the elite chromosome, a copy of the elite chromosome replaces the worst chromosome in the population. Elitism is important because it guarantees that the quality of the best solutions found over generations is monotonically nondecreasing.

1.4.5 Crossover Operators

The crossover operator selects a random number of pairs of scheduling strings, where every string has an equal probability of being selected. For each pair, it randomly generates a cut-off point, which divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative positions of these subtasks in the other original scheduling string in the pair, thus guaranteeing that the newly generated scheduling strings (which replace the originals) are valid schedules. Figure 1.4 demonstrates such a scheduling string crossover process.

The crossover operator selects a random number of pairs of matching strings, where every string has an equal probability of being selected. For each pair, it randomly generates a cut-off point to divide both matching strings of

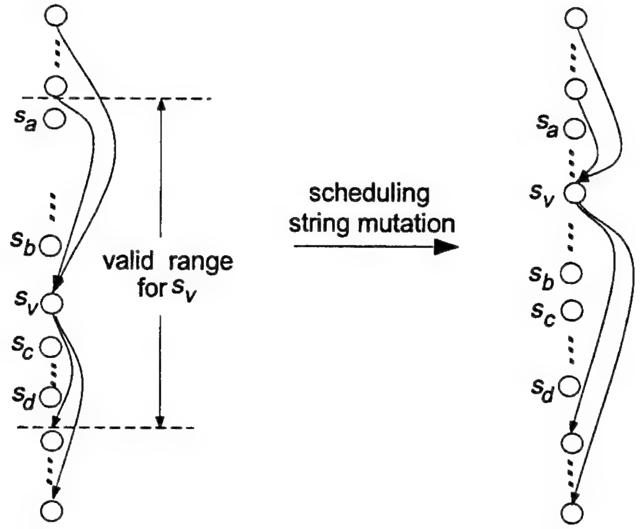


Fig. 1.5 A scheduling string mutation example. Only edges to and from the victim subtask s_v are shown. Before the mutation, s_v is between s_b and s_c. After the mutation, it is moved to between s_a and s_b.

the pair into two parts. Then the machine assignments of the bottom parts are exchanged, and the new strings replace the originals.

1.4.6 Mutation Operators

The scheduling string mutation operator selects a random number of scheduling strings, where every string has an equal probability of being selected. Then for each chosen scheduling string, it randomly selects a victim subtask. The valid range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed without violating any data dependency constraints. Specifically, the valid range is after all source subtasks of the victim subtask and before any destination subtask of the victim subtask. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its valid range. The new string replaces the original. Figure 1.5 shows an example of this mutation process. For the matching string mutation operator, every matching string has an equal probability of having one of its subtasks assigned to a different, randomly selected, machine.

1.4.7 Evaluation

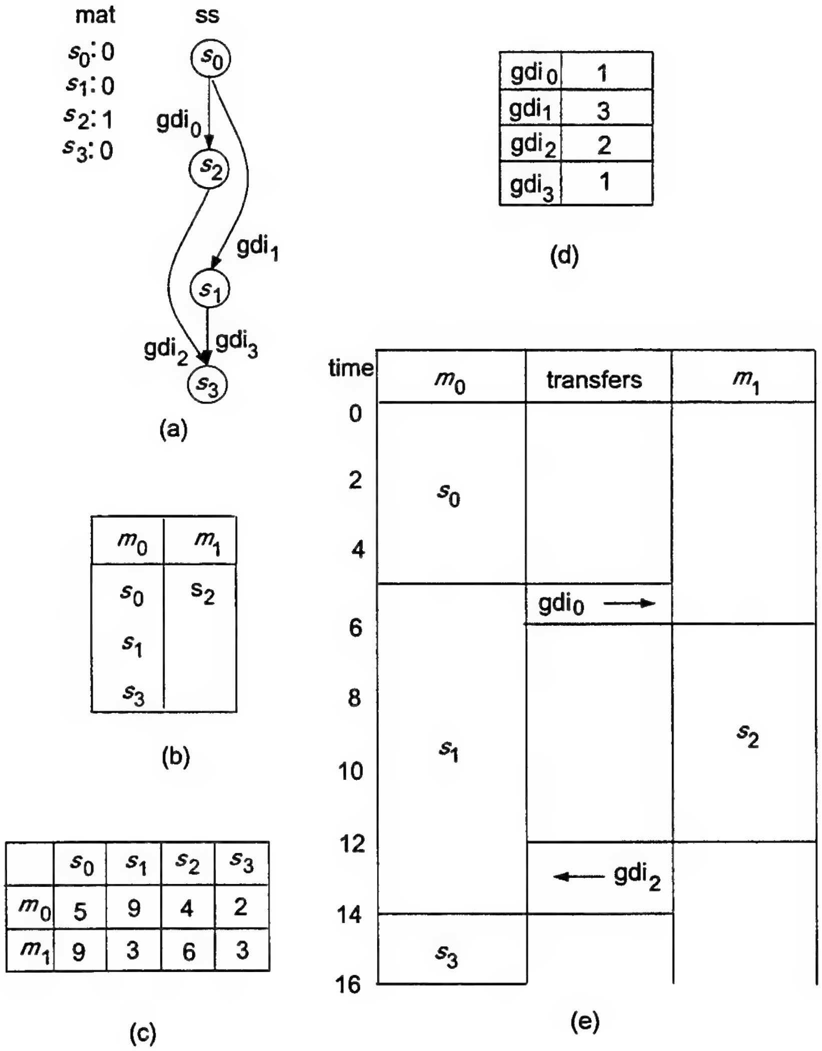
The final step of an iteration of a GA is the evaluation of each chromosome to determine its fitness value. In this GA-based approach, the chromosome structure is independent of any particular communication subsystem. Only the evaluation step needs the communication characteristics of the given HC system to schedule the data transfers.

To test the effectiveness of this GA-based approach, an example communication system modeled after a HiPPI LAN with a central crossbar switch [28, 53] is assumed to connect a suite of machines. Each machine in the HC suite has one input data link and one output data link. All these links are connected to a central crossbar switch. If a subtask needs a global data item that is produced or consumed earlier by a different subtask on the same machine, the communication time for this item is zero. Otherwise, the communication time is obtained by dividing the size of the global data item by the smaller bandwidth of the output link of the source machine and the input link of the destination machine. In this research, it is assumed that for a given machine, the bandwidths of the input link and the output link are equal to each other. It is also assumed that the crossbar switch has a higher bandwidth than that of each link. The communication latency between any pair of machines is assumed to be the same. Data transfers are neither preemptive nor multiplexed. Once a data transfer path is established, it cannot be relinquished until the data item (i.e., some gdi_k) scheduled to be transferred over this path is received by the destination machine. Multiple data transfers over the same path must be serialized.

In the evaluation step, for each chromosome the final order of execution of the subtasks and the inter-machine data transfers are determined. The evaluation procedure considers the subtasks in the order they appear on the scheduling string. Subtasks assigned to the same machine are executed exactly in the order specified by the scheduling string. For subtasks assigned to different machines, the actual execution order may deviate from that specified by the scheduling string due to factors such as input-data availability and machine availability. This is explained below.

Before a subtask can be scheduled, all of its input global data items must be received. For each subtask, its input data items are considered by the evaluation procedure in the order of their producers' relative positions in the scheduling string. In Figure 1.6, a simple example is shown to illustrate the evaluation for a given chromosome. In this example (as well as some others given later), because there are only two machines, the source and destination machines for the gdi transfers are implicit.

When a subtask to be scheduled has multiple input gdis that have not been received, the global data item whose producer subtask is listed earliest in the scheduling string is considered first. The reason for this ordering is to attempt to better utilize the overlap of subtask executions and inter-machine data communications. The following example illustrates this idea. Let ss(0)



(a)

m_0	m_1
s_0	s_2
s_1	
s_3	

(b)

	s_0	s_1	s_2	s_3
m_0	5	9	4	2
m_1	9	3	6	3

(c)

gdi_0	1
gdi_1	3
gdi_2	2
gdi_3	1

(d)

(e)

Fig. 1.6 A very simple example showing the evaluation step: (a) the chromosome; (b) the subtask execution ordering on each machine given by (a); (c) the estimated subtask execution times; (d) the gdi inter-machine transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution and data transfer timings, where the completion time for this chromosome is 16.

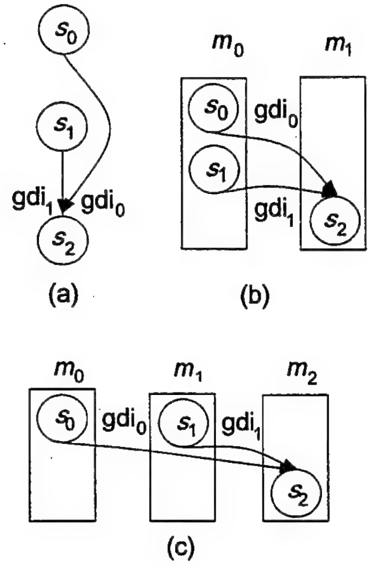


Fig. 1.7 An example scheduling string with two possible matching strings: (a) the example scheduling string; (b) the situation when the source subtasks of the input gdis are assigned to the same machine; (c) the situation when the source subtasks of the input gdis are assigned to different machines.

= s_0, ss(1) = s_1, and ss(2) = s_2, as shown in Figure 1.7(a). Let s_2 need two gdis, gdi$_0$ and gdi$_1$, from s_0 and s_1, respectively. Depending on the subtask to machine assignments, the data transfers of gdi$_0$ and gdi$_1$ could be either local, i.e., within a machine, or across machines. If at least one data transfer is local, then the scheduling is trivial because it is assumed that local transfers within a machine take negligible time. However, there exist two situations where both data transfers are across machines so that they need to be ordered.

Situation 1: Let s_0 and s_1 be assigned to the same machine m_0 and s_2 be assigned to another machine m_1, as shown in Figure 1.7(b). In this situation, because s_0 is to be executed before s_1, gdi$_0$ is available before gdi$_1$ becomes available on machine m_0. Thus, it is better to consider the gdi$_0$-transfer before the gdi$_1$-transfer.

Situation 2: Let the three subtasks s_0, s_1, and s_2 be assigned to three different machines m_0, m_1, and m_2, as shown in Figure 1.7(c). In this situation, if there is a data dependency from s_0 to s_1, then s_0 finishes its execution before s_1 could start. Therefore, gdi$_0$ is available before gdi$_1$ becomes available. Hence, it is better to consider the gdi$_0$-transfer before the gdi$_1$-transfer. If there are no data dependencies from s_0 to s_1, the gdi$_0$-transfer can still be considered before the gdi$_1$-transfer. While this may not be necessary in this case, it is reasonable to do because there may be some other chromosome(s) that have ss(0) = s_1 and ss(1) = s_0. When such a chromosome is evaluated, the gdi_1-transfer will be considered before the gdi_0-transfer. Therefore, it is possible for all input gdi scheduling orderings for gdi_0 and gdi_1 to be examined.

Data forwarding is another important feature of this evaluation process. For each input data item to be considered, the evaluation process chooses the source subtask from among the producer of this data item and all the consumers that have received this data item. These consumers are forwarders [51]. The one (either the producer or a forwarder) from which the destination subtask will receive the data item at the earliest possible time is chosen.

After the source subtask is chosen, the data transfer for the input data item is scheduled. A transfer starts at the earliest point in time when the path from the source machine to the destination machine is free for a period that is at least equal to the needed transfer time. Thus, it is possible that, for example, gdi_1 is considered before gdi_2, but gdi_2 is transferred before gdi_1. This is referred to as out-of-order scheduling of data transfers because the data transfers do not occur in the order in which they are considered (i.e., the in-order schedule). This (possibly) out-of-order scheduling of the input item data transfers utilizes previously idle bandwidths of the communication links, and thus could make some input data items available to some subtasks earlier than the in-order scheduling. As a result, some subtasks could start their execution earlier, which would in turn decrease the overall task completion time. Figures 1.8 and 1.9 show the in-order scheduling and the out-of-order scheduling for the same chromosome, respectively. In the in-order scheduling, the transfer of gdi_1 is scheduled before the transfer of gdi_2 because subtask s_2's input data transfers are considered before those of subtask s_3. In this example, the out-of-order schedule does decrease the total execution time of the given task.

When two chromosomes have different matching strings, they are different solutions because the subtask-to-machine assignments are different. However, two chromosomes that have the same matching string but different scheduling strings may or may not represent the same solution. This is because the scheduling string information is used in two cases: (1) for scheduling subtasks that have been assigned to the same machine and (2) for examining data transfers. Two different scheduling strings could result in the same ordering for (1) and (2).

After a chromosome is evaluated, it is associated with a fitness value, which is the time when the last subtask finishes its execution. That is, the fitness value of a chromosome is the overall execution time of the task, given the mapping decision specified by this chromosome and by the evaluation process.

In summary, this evaluation mechanism considers subtasks in the order in which they appear in the scheduling string. For a subtask that requires some gdis from other machines, the gdi-transfer whose producer subtask appears earliest in the scheduling string is examined first. When scheduling a gdi-

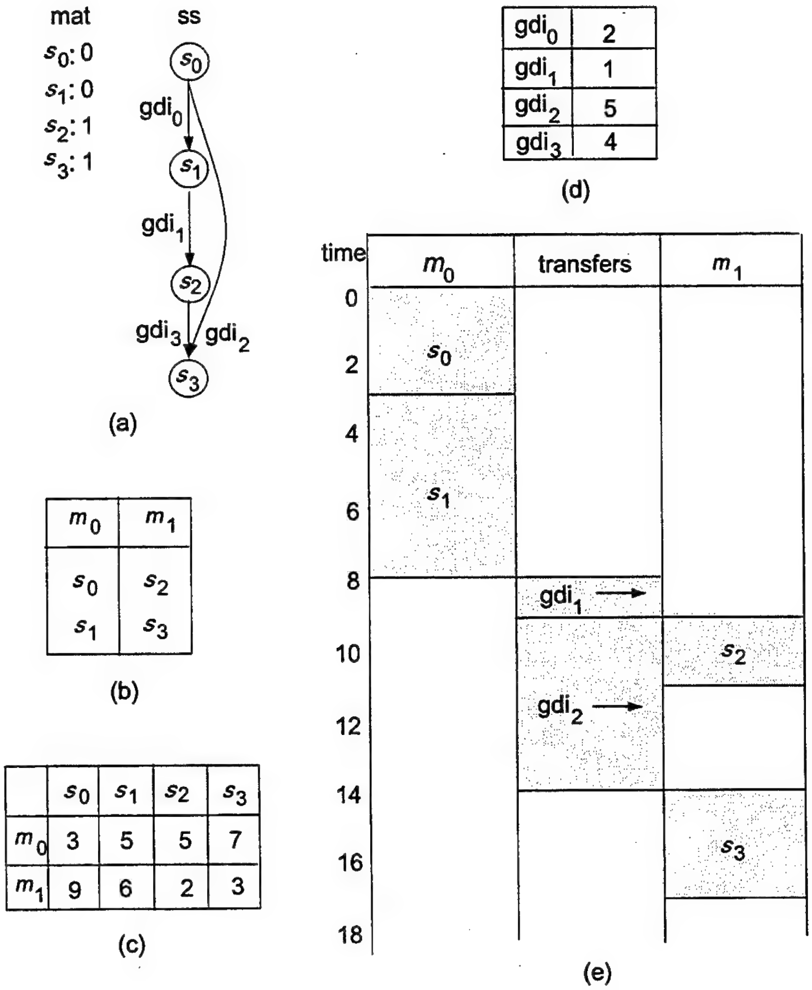


Fig. 1.8 An example showing the in-order scheduling of a chromosome: (a) the chromosome; (b) the subtask execution ordering on each machine; (c) the estimated subtask execution times; (d) the gdi transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution and data transfer timings using in-order transfers (the gdi_1-transfer occurs before the gdi_2-transfer), where the completion time is 17.

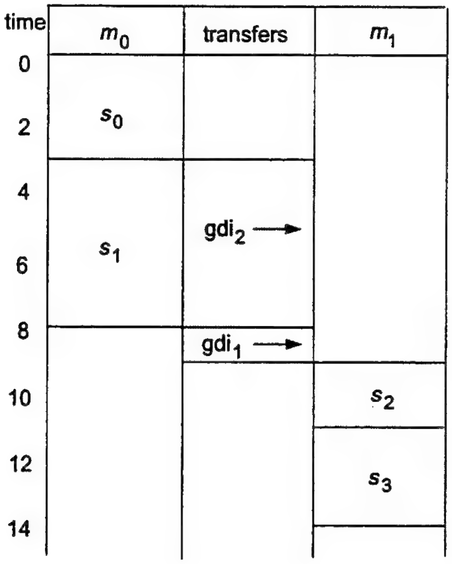


Fig. 1.9 An example showing the the out-of-order scheduling, where the chromosome and other statistics are the same as in Figure 1.8. The completion time is 14.

transfer, both the producing and the forwarding subtasks are considered as possible sources. The source subtask that lets this consumer subtask receive this gdi at the earliest possible time is chosen to send the gdi. The out-of-order scheduling of the gdi transfers over a path could further reduce the completion time of the application.

1.4.8 Experimental Results

To measure the performance of this GA-based approach, randomly generated scenarios were used, where each scenario corresponded to an SPDAG, the associated subtask execution times, the sizes of the associated global data items, and the communication link bandwidths of the machines. The scenarios were generated for different numbers of subtasks and different numbers of machines, as specified below. The estimated expected execution time for each subtask on each machine, the number of global data items, the size of each global data item, and the bandwidth of each input link of each machine were randomly generated with uniform probability over some predefined ranges. For each machine, the bandwidth of the output link is made equal to that of the input link. The producer and consumers of each global data item were also generated randomly. The scenario generation used a $|G| \times |S|$ dependency matrix to guarantee that the precedence constraints from data dependencies were acyclic.

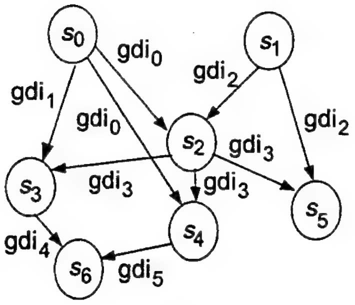
These randomly generated scenarios were used for three reasons: (1) it is desirable to obtain data that demonstrate the effectiveness of the approach over a broad range of conditions, (2) a generally accepted set of HC benchmark tasks does not exist, and (3) it is not clear what characteristics a "typical" HC task would exhibit [56]. Determining a representative set of HC task benchmarks remains a challenge for the scientific community in this research area.

In this work, small-scale and larger scenarios were used to quantify the performance of this GA-based approach. The scenarios were grouped into three categories, namely tasks with light, moderate, and heavy communication loads. A lightly communicating task has its number of global data items in the range of $0 \leq |G| < (1/3)|S|$; a moderately communicating task has its number of global data items in the range of $(1/3)|S| \leq |G| < (2/3)|S|$; and a heavily communicating task has its number of global data items in the range of $(2/3)|S| \leq |G| < |S|$. The ranges of the global data item sizes and the estimated subtask execution times were both from 1 to 1,000. For these scenarios, the bandwidths of the input and output links were randomly generated, ranging from 0.5 to 1.5. Hence, the communication times in these scenarios were source and destination machine dependent.

The probability of crossover was the same for the matching string and the scheduling string. The probability of mutation was also the same for the matching string and the scheduling string. The stopping criteria were (1) the number of iterations had reached 1000, (2) the population had converged (i.e., all the chromosomes had the same fitness value), or (3) the currently best solution had not improved over the last 150 iterations. All the GA runs discussed in this section stopped due to their best solutions not improving for 150 iterations.

The GA-based approach was first applied to 20 small-scale scenarios that involved up to ten subtasks, three machines, and seven global data items. The GA runs for small-scale scenarios had the following parameters. The probabilities for scheduling string and matching string crossovers, were both chosen to be 0.4, and scheduling string and matching string mutations were both chosen to be 0.1. The GA population size, P, for small-scale scenarios was chosen to be 50. The angle ratio of the sectors on the roulette wheel for two adjacently ranked chromosomes, R, was chosen to be $1 + 1/P$. By using this simple formula, the angle ratio between the slots of the best and median chromosomes for $P = 50$ (and also for $P = 200$ for larger scenarios discussed later in this section) was very close to the optimal empirical ratio value of 1.5 in [60].

The results from a small-scale scenario are used here to illustrate the search process. This scenario had $|S| = 7$, $|M| = 3$, and $|G| = 6$. The SPDAG, the estimated execution times, and the transfer times of the global data items are shown in Figure 1.10. The total numbers of possible different matching



(a)

	m_0m_1	m_0m_2	m_1m_2
gdi$_0$	489	321	489
gdi$_1$	1244	818	1244
gdi$_2$	62	41	62
gdi$_3$	830	545	830
gdi$_4$	387	255	387
gdi$_5$	999	656	999

(c)

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
m_0	872	251	542	40	742	970	457
m_1	898	624	786	737	247	749	451
m_2	708	778	23	258	535	776	15

(b)

Fig. 1.10 A small-scale simulation scenario: (a) the SPDAG, (b) the estimated execution times, and (c) the transfer times of the global data items from a given source machine to a given destination machine.

strings and different valid scheduling strings (i.e., topological sorts of the SPDAG) were $3^7 = 2187$ and 16, respectively. Thus, the total search space had $2187 \times 16 = 34992$ possible chromosomes.

Exhaustive searches were performed to find the optimal solutions for the small-scale scenarios. For each of the small-scale scenarios that were conducted, the GA-based approach found one or more optimal solutions that had the same completion time, verified by the best solution(s) found by the exhaustive search. The GA search for a small-scale scenario that had ten subtasks, three machines, and seven global data items took about one minute to find multiple optimal solutions on a Sun Sparc5 workstation while the exhaustive search took about eight hours to find these optimal solutions.

The performance of this GA-based approach was also examined using larger scenarios with up to 100 subtasks and 20 machines. These larger scenarios were generated using the same procedure as for generating the small scenarios, except that the GA population size for larger scenarios was chosen to be 200.

Larger scenarios are intractable problems. It is currently impractical to directly compare the quality of the solutions found by the GA-based approach for these larger scenarios with those found by exhaustive searches.

It is also often difficult to compare the performance of different HC task mapping approaches due to the different HC system models that are assumed by researchers when they design mapping heuristics. However, the model used in [30] is similar to the one being used in this research work. Hence, the performance of the GA-based approach on larger scenarios was compared with the non-evolutionary levelized min-time (LMT) heuristic proposed in [30]. (In Section 1.6, a variety of heuristics are compared by adapting them for a simple common model.)

The LMT heuristic first establishes levels of subtasks in the following way. The subtasks that have no input global data items are at the highest level. Each of the remaining subtasks is at one level below the lowest producer of its global data items. The subtasks at the highest level are to be considered first. The LMT heuristic averages the estimated execution times for each subtask across all machines. At each level, a level-average execution time, i.e., the average of the machine-average execution times of all subtasks at this level, is also computed. If there are some levels between a subtask and its closest child subtask, the level-average execution time of each middle level is subtracted from the machine-average execution time of this subtask. The adjusted machine-average execution times of the subtasks are used to determine the priorities of the subtasks within each level, i.e., a subtask with a larger average is to be considered earlier at its level. If the number of subtasks at a level is greater than the number of machines in the HC suite, the subtasks with smaller averages are merged so that as a result, the number of combined subtasks at each level equals the number of machines available. When a subtask is being considered, it is assigned to the fastest machine available from those machines that have not yet been assigned any subtasks from the same level.

Another non-evolutionary heuristic, the baseline (BL), was developed as part of this GA research and the solution it found was incorporated into the initial population. Similar to the LMT heuristic, the baseline heuristic first establishes levels of subtasks based upon their data dependencies. Then all subtasks are ordered such that a subtask at a higher level comes before one at a lower level. The subtasks in the same level are arranged in descending order of their number of output global data items (ties are broken arbitrarily). The subtasks are then scheduled as follows. Let the i-th subtask in this order be $\underline{\sigma_i}$, where $0 \leq i < |S|$. First, subtask σ_0 is assigned to a machine that gives the shortest completion time for σ_0. Then, the heuristic evaluates $|M|$ assignments for σ_1, each time assigning σ_1 to a different machine, with the previously decided machine assignment of σ_0 left unchanged. The subtask σ_1 is finally assigned to a machine that gives the shortest overall completion time for both σ_0 and σ_1. The baseline heuristic continues to evaluate the remaining subtasks in the order defined above. When scheduling subtask σ_i, $|M|$ possible machine assignments are evaluated, each time with the previously decided machine assignments of subtasks σ_j $(0 \leq j < i)$ left unchanged. Subtask σ_i is finally assigned to a machine that gives the shortest overall completion time of subtasks σ_0 through σ_i. The total number of evaluations is thus $|S| \times |M|$, and only i subtasks (out of $|S|$) are considered when performing evaluations for the $|M|$ machine assignments for subtasks σ_i.

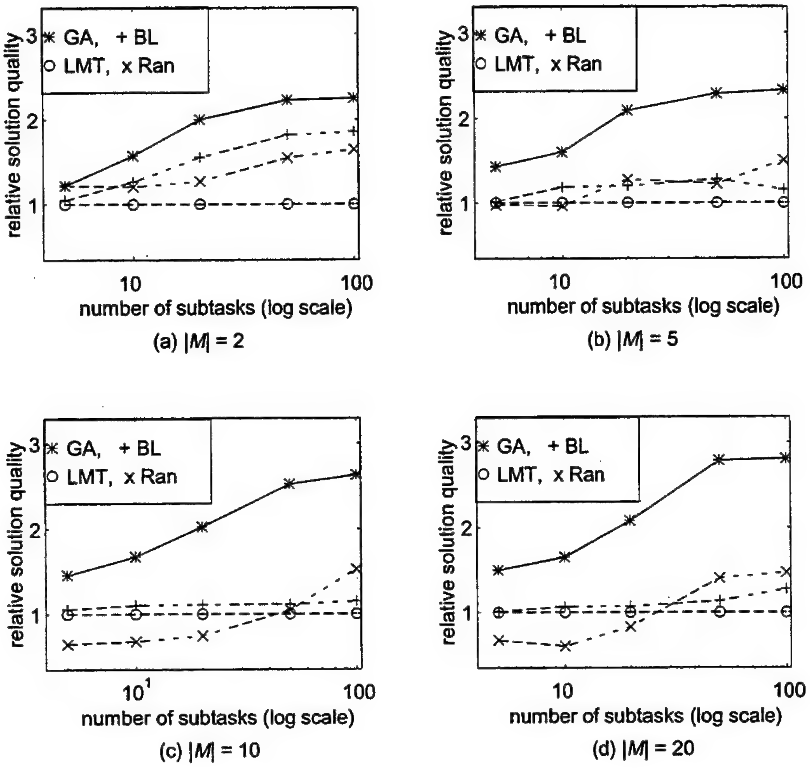
To determine the best GA parameters for solving larger HC mapping problems, 50 larger scenarios were randomly generated in each communication category. Each of these scenarios contained 50 subtasks and five machines. For each scenario, GA runs were conducted for the following combinations of crossover probability and mutation probability. The crossover probability ranged from 0.1 to 1.0 in steps of 0.1, and the mutation probability ranged from 0.04 to 0.40 in steps of 0.04 and from 0.4 to 1.0 in steps of 0.1. Let the relative solution quality be the task completion time of the solution found by the LMT heuristic divided by that found by the approach being investigated. A greater value of the relative solution quality means that the approach being investigated finds a better solution to the HC mapping problem (i.e., with a shorter overall completion time for the application task represented by the SPDAG). With each crossover and mutation probability pair and for each communication load, the average relative solution quality of the 50 GA runs, each on a different scenario, was computed.

For each communication load category, a region of good performance could be identified for a range of crossover and mutation probabilities. The regions of good performance generally consisted of moderate to high crossover probability and low to moderate mutation probability. This is consistent with the results from the GA literature, which show that crossover is GA's major operator and mutation plays a secondary role in GA searches [14, 25, 49].

The crossover and mutation probabilities with which the best relative solution quality had been achieved were used in each corresponding communication load category. When mapping real tasks, the communication load can be determined by computing the ratio of the number of global data items to the number of subtasks. Once the communication load category is known, a probability pair from the corresponding region of good performance can be used.

On Sun Sparc5 workstations, for these larger scenarios, both the LMT heuristic and the baseline heuristic took no more than one minute of CPU time to execute. The average CPU execution time of the GA-based approach on these scenarios ranged from less than one minute for the smallest scenarios (i.e., five subtasks, two machines, and light communication load) to about three and one half hours for the largest scenarios (i.e, 100 subtasks, 20 machines, and heavy communication load). Recall that it is assumed that this GA-based approach will be used for application tasks that are large production jobs such that the one-time investment of this high execution time is justified.

The performance of the GA-based approach was also compared with that of a random search. For each iteration of the random search, a chromosome was randomly generated, this chromosome was evaluated, and the fitness value was compared with the saved best fitness value. If the fitness value of the current



(a) |*M*| = 2

(b) |*M*| = 5

(c) |*M*| = 10

(d) |*M*| = 20

Fig. 1.11 Performance comparisons of the GA-based approach with a crossover probability of 1.0 and a mutation probability of 0.2, relative to the LMT heuristic for heavily communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic and the random search are also shown.

chromosome was better than the saved best value, it became the saved best fitness value. For each scenario, the random search iterated for the same length of time as that taken by the GA-based approach on the same scenario.

Figure 1.11 shows the performance comparisons between the LMT heuristic and the GA-based approach for heavily communicating larger scenarios. The GA-based approach used a crossover probability of 1.0 and a mutation probability of 0.2. These values were the best performing probabilities over a wide range of combinations examined for a separate set of test scenarios. In the figure, the horizontal axes are the number of subtasks in log scale. The vertical axes are the relative solution quality of the various approaches. Each point in the figure is the average of 50 independent scenarios. The performance comparisons among the GA-based approach, the LMT heuristic, the baseline heuristic, and the random search for moderately communicating and lightly communicating larger scenarios can be found in [58].

In all cases, the GA-based approach presented here outperformed the BL and LMT heuristics and the random search. The improvement of the GA-based approach over the others showed an overall trend to increase as the number of subtasks increased. The exact shape of the GA-based-approach performance curves is not as significant as the overall trends because the curves are for a heuristic operating on randomly generated data, resulting in some varied performance even when averaged over 50 scenarios for each data point.

1.4.9 Summary

A novel genetic-algorithm-based approach for task mapping in HC environments was presented. This GA-based approach can be used in a variety of HC environments because it does not rely on any specific communication subsystem model. It is applicable to the static scheduling of production jobs and can be readily used for scheduling multiple independent tasks (and their subtasks) collectively. For small-scale scenarios, the proposed approach found optimal solutions. For larger scenarios, it outperformed two non-evolutionary heuristics and a random search.

There are a number of ways this GA-based approach for HC task mapping may be built upon for future research. These include extending this approach to allow multiple producers for each of the global data items, parallelizing the GA-based approach, developing evaluation procedures for other communication subsystems, and considering loop and data-conditional constructs that involve multiple subtasks.

1.5 SEMI-STATIC MATCHING AND SCHEDULING OF SUBTASKS

1.5.1 Overview

In this section, the application domain and HC platform assumed for the semi-static approach are described [7, 8]. This is followed by a discussion of the enhancements made to the GA described in Section 1.4 for determining off-line mappings in this situation. Finally, Subsection 1.5.7 summarizes the results of an extensive performance study of a simulated semi-static mapping system [35].

1.5.2 Application Domain: Automatic Target Recognition

One type of automatic target recognition (ATR) system takes a stream of images from a group of sensors and produces a scene description [54], tracking the movement of possible targets through the sequence of images. A simplified example of an ATR task can be found in [35]. The various types of image processing elements required in an ATR system can be broadly classified into three groups: low-level processing (numeric computation), intermediate-level processing (quasi-symbolic computation, e.g., where numeric and symbolic types of operations are used to describe surfaces and shapes of objects in the scene), and high-level processing (symbolic computation, e.g., used to produce the scene description) [1, 4]. Each of these subtasks may allow the use of multiple processors of the same type for efficient execution. Heterogeneous parallel architectures consisting of multiple copies of different types of processors (e.g., SHARC DSP and PowerPC RISC processors [16]) are appropriate computing platforms for efficiently handling computational tasks with such diverse requirements.

The class of ATR applications considered can be modeled as an iterative execution (one execution for each image in the stream) of a set of partially ordered subtasks. Thus, the ATR task can be modeled by a DAG in which the nodes represent subtasks and the edges represent the communications among subtasks. It is assumed that an ATR application in this class is a production job that is executed repeatedly. Therefore, it is worthwhile to invest extra off-line time in preparing an effective mapping of the application onto the hardware platform used to execute it.

1.5.3 An Intelligent Operating System

A major distinctive architectural feature of the envisioned IOS is the capability for the on-line use of off-line computed mappings [7, 8, 35]. The ATR Kernel makes decisions on how a given ATR application task should be accomplished, including determining the partial ordering of subtasks and which algorithms could be used to accomplish each subtask. The HC Kernel uses a semi-static method to decide how the partially-ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. Furthermore, the HC Kernel interacts with the Basic Kernel (the low-level operating system) to initiate the application and monitor its execution. This allows the HC Kernel to decide at the end of each iteration through the application if the subtasks should be remapped onto the hardware platform. Information from the Algorithm Database and the Knowledge Base is used to support the ATR and HC Kernels.

The Algorithm Database includes one or more implementations of each algorithm (e.g., one for each processor type). The Algorithm Database also contains the expected execution time of each algorithm implementation, typically specified as a function of type and number of processors assigned, interprocessor communication time, and certain input data and application characteristics. Dynamic parameters are those input characteristics, such as amount of clutter, and application characteristics, such as number of located objects to be identified, that: (1) will change during run time, (2) can be computed by the application as it executes, and (3) may impact the execution time of each subtask in the task graph. This is an expected time, rather than a definite time, because it may vary depending on the actual values of the input data being processed. System parameters, such as the number of each type of processors in the platform, are stored in the Knowledge Base. The off-line and on-line components of the HC Kernel are detailed below. For more details about the IOS, the reader is referred to [8].

1.5.4 Off-line HC Kernel Component: MDDG Generator

A mapped data dependency graph (MDDG) is a DAG annotated with the task-to-machine mapping. The HC Kernel MDDG Generator, which is a major component of the off-line IOS, is responsible for mapping each DAG onto the heterogeneous parallel hardware platform creating a corresponding MDDG. The structure of the HC Kernel MDDG Generator is such that any effective heuristic could be employed. For the iterative ATR application domain, it is possible to use off-line precomputed mappings to reconfigure resources in real time.

Because different sets of dynamic parameter values can lead to different subtask execution times (e.g., more objects to be recognized in the scene), the HC Kernel MDDG Generator will, in general, generate different mappings for the same DAG (corresponding to different sets of dynamic parameter values). It is assumed that the ranges of the dynamic parameters are known. The space of dynamic parameters is partitioned into a number of disjoint regions, and within each region a random set of representative dynamic parameter vectors are chosen, each of which is called a sample vector. For each sample vector, an off-line mapping heuristic (the enhanced GA described in Subsection 1.5.6) uses the following information to create a mapping, i.e., to transform a DAG into an MDDG: (1) the structure of the underlying DAG; (2) the expected execution time of each subtask on a set of processors (of the same type) assigned, as a function of the type and number of processors; (3) the inter-subtask data transfers needed, in terms of formats and expected sizes of the data items to be transferred; (4) the expected time to send data from one processor to another as a function of the size of the data item to be transferred; and (5) the number of each type of processor that is in the hardware platform.

The mapping for each sample vector is exhaustively evaluated for every other sample vector in the region by applying the mapping to the DAG and computing the task execution time using that other sample vector's dynamic parameter values. The mapping that gives the minimum average execution time is chosen as the representative mapping for the corresponding region in the dynamic parameter space. This representative mapping and the corresponding average execution time are stored in the off-line mapping table (i.e., the MDDG Table), which is a multi-dimensional array, indexed by dynamic parameter ranges. Thus, for a given DAG, the MDDG for each region of the dynamic parameter space is stored in the MDDG Table for that DAG, along with the corresponding expected execution time.

1.5.5 On-line HC Kernel Component: Execution Monitor

The HC Kernel Monitor is an on-line component responsible for (1) establishing the initial mapping of the given application onto the hardware platform, and (2) monitoring the execution of the application and, at the end of each iteration through the corresponding DAG, deciding if and how the mapping of the application onto the hardware platform should be changed based on information about the actual values of the dynamic parameters. Examples of dynamic parameters include the contrast level of an image, the number of objects in a scene, and the average size (in pixels) of an object in a scene.

The initial mapping is selected from the MDDG Table for that DAG based on menu-driven input from the application user about an initial value to assume for each of the dynamic parameters. During execution of the application, the HC Kernel Monitor receives updated actual values of the dynamic parameters at the end of each iteration through the corresponding DAG. The HC Kernel Monitor will use the most recent values of these dynamic parameters to estimate if changing the mapping will reduce the expected execution time of the next iteration through the corresponding DAG. It does this by comparing the actual execution time of the last completed iteration with the sum of an estimated time for reconfiguration plus the expected execution time from the MDDG Table entry for the region that includes the most recent dynamic parameter values known. Thus, this decision is made in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration. If it is desirable to change the mapping, then the HC Kernel Monitor will pass the MDDG Table entry index to the Basic Kernel for use in the next iteration. If not, the same mapping will continue to be used.

1.5.6 Genetic Algorithm Used for Semi-Static Approach

The genetic algorithm described in Section 1.4 is enhanced for determining the off-line mappings. Because each subtask can be mapped to multiple processors of the same type, a new string, called the allocation string, is also incorporated into each chromosome in the enhanced GA. Specifically, each entry i in the allocation string represents the number of processors of a certain type (specified by the corresponding entry in the matching string) assigned to the subtask s_i. Typically, multiple subtasks will be assigned to some of the same processors in a processor group. The subtasks are then executed in a non-preemptive manner based on the ordering that is specified by the scheduling string.

The initial allocation string for each chromosome is generated by randomly selecting a value from 1 to p_{opt} (defined in Subsection 1.5.7.2) as the number of processors allocated to each subtask (for the processor group type specified in the matching string). The solution generated by a fast heuristic is also included in the initial population. The heuristic used is a fast static scheduling algorithm, called the Earliest Completion Time (ECT) algorithm [57], which is similar to the LMT heuristic discussed in Subsection 1.4.8 and is described in [35]. One chromosome with an allocation string where each entry is the optimal number of processors for that subtask (individually) is also included in the initial population.

The mutation operator for the allocation string randomly selects an entry in the string and locally optimizes it by changing the number of processors to a value that gives the best total task execution time. The crossover operator for allocation strings is the same as the one used for matching strings. The probability for mutation and crossover for all strings in the experiments below is 0.4.

The GA is executed ten times for each sample vector. To enhance diversity, only five of the ten runs include a chromosome generated by the ECT algorithm in the initial population.

1.5.7 Performance Results

1.5.7.1 Overview In [34, 35], an extensive performance study was performed on a simulated HC Kernel implemented using the enhanced GA as the off-line heuristic. The goal of the study was to evaluate the ideas underlying the particular semi-static method of on-line use of off-line derived mappings described above (referred to as On-Off in subsequent sections). Four approaches were compared in the experiments: (1) the On-Off approach; (2) the ECT algorithm as a dynamic scheduling algorithm; (3) the infeasible approach of using the GA as a dynamic scheduling algorithm (referred to as GA On-line); and (4) an ideal but impossible approach which uses the GA on-line with the exact (as yet unknown) dynamic parameters for the iteration to be executed next (referred to as Ideal). In the latter two schemes, a mapping determined for a previous iteration is also included in the initial population of the current iteration, and reconfiguration times are ignored. These two infeasible schemes were merely used as references for comparison to solution quality of the former two approaches.

1.5.7.2 System Model To evaluate the On-Off semi-static mapping methodology, an example architecture was chosen [35]. It consisted of four different types of processors with 16 processors for each type (i.e., total number of processors in the HC platform is 64). The processors within each type are connected via a 17-port crossbar switch, whereas the four different type of processors are connected using a four-port crossbar switch. The execution time of a subtask and the communication time between subtasks in an application task were modeled by equations that are functions of the dynamic parameters. Specifically, the simple execution time expression used in this task model is a version of Amdahl's law extended by a term representing the parallelization overhead (e.g., synchronization and communication). The serial and parallel fractions of a subtask are frequently represented using similar models (e.g., [9, 43]). The execution time expression for subtask s_i includes: (1) three generic dynamic parameters, α, β, and γ, (2) the number of processors used, p, and (3) three coefficients, a_i, b_i, and c_i. The parallel fraction and serial fraction of subtask s_i are represented by $a_i\alpha/p$ and $c_i\gamma$, respectively. The parallelization overhead is represented by $b_i\beta\log p$. The relative speed of a subtask s_i on a processor of type u is given by the heterogeneity factor h_{iu}. The execution time of subtask s_i is then given by the expression: $h_{iu}(a_i\alpha/p + b_i\beta\log p + c_i\gamma)$. By differentiating this equation and equating it to zero, the optimal value of p (p_{opt}) that leads to the minimum execution time for a given subtask is $(a_i\alpha)/(b_i\beta)$.

It is assumed that the size of the data to be transferred between two subtasks s_i and s_j consists of a fixed portion d_{ij} and a variable portion $e_{ij}\mu$, where μ is a fourth dynamic parameter. For communication between virtual machines whose processor types are u and v, S_{uv} and R_{uv} are the message start-up time and the data transmission rate, respectively. The inter-subtask communication time C_{uv} between subtask s_i on a virtual machine with processors of type u and subtask s_j on a virtual machine with processors of type v is given by the expression: $S_{uv} + (d_{ij} + e_{ij}\mu)/R_{uv}$.

The example architecture and the simplified generic equations are used as input to the simulated HC Kernel only. The On-Off method can be adopted for other target architectures and the actual time equations specified by the application developer. In practice, a particular single dynamic parameter can impact any subset of the components of a given subtask's execution time equation.

1.5.7.3 Workload To investigate the performance of the On-Off approach, randomly generated DAGs containing 10, 50, 100, or 200 nodes were used. These DAGs included regularly structured graphs (in-trees, out-trees, and fork-joins [34, 35]) and randomly structured DAGs. For each size and structure, ten different graphs were generated and, thus, a total of 160 graphs were used. The input to the simulated on-line module consists of an execution profile that comprises a certain number of iterations of executing the DAG. Examples of two randomly generated execution profiles containing 20 iterations are shown in Table 1.1. In each profile, the dynamic parameter values change from one iteration to another within certain ranges (α: [1000–5000], β: [5–25], γ: [100–500], and μ: [20–100]). Specifically, the average percentage change in dynamic parameter values in Profile A and Profile B are 5% and 40%, respectively. In each profile, row i represents the values of the dynamic parameters observed after execution of the DAG for iteration i is finished. Thus, when execution of the task iteration begins, the on-line module does not know the (simulated) actual values of the dynamic parameters for that iteration. The on-line module has to determine a mapping for iteration i

iteration	α	β	γ	μ
0	3000	15	300	60
1	2821	15	287	63
2	2949	12	302	65
3	3073	12	286	68
4	3228	11	273	71
5	3090	13	258	67
6	3256	11	272	70
7	3424	16	259	73
8	3621	16	271	75
9	3811	13	260	78
10	4014	17	245	81
11	4229	13	257	77
12	3994	19	242	80
13	4179	15	253	83
14	4386	15	264	78
15	4208	13	249	82
16	4016	14	236	77
17	3835	16	226	81
18	4026	19	238	84
19	4258	16	251	88
20	4479	15	265	92

(a) Profile *A*

iteration	α	β	γ	μ
0	3000	15	300	60
1	4309	15	409	82
2	2635	7	268	43
3	3894	6	361	27
4	2241	8	197	39
5	1265	12	287	52
6	1699	16	420	75
7	1138	11	282	50
8	1543	12	153	67
9	2205	17	225	97
10	3198	10	332	51
11	4678	18	477	73
12	2588	8	315	48
13	1358	16	211	67
14	1794	17	307	98
15	2605	11	163	61
16	3719	17	240	87
17	2478	9	332	53
18	1507	16	466	76
19	2081	8	243	50
20	3053	17	149	70

(b) Profile *B*

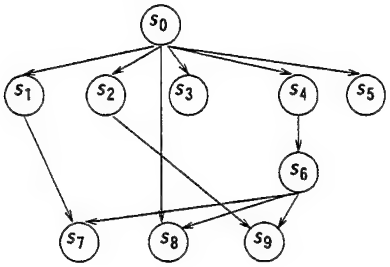
Table 1.1 Execution profiles of dynamic parameters: (a) Profile *A* (average percentage change in dynamic parameter values = 5%) and (b) Profile *B* (average percentage change = 40%).

based on the dynamic parameter values of iteration $i - 1$. The methods for generating the DAGs and execution profiles can be found in [34, 35].

In generating the off-line mapping tables, each dynamic parameter range is partitioned into four equal intervals, creating $4^4 = 256$ regions. Thus, the MDDG Table stores 256 mappings. Ten sample vectors are randomly chosen from each region. Because the GA for each sample vector is performed with ten different initial populations, the GA is executed $256 \times 10 \times 10 = 25600$ times to build each MDDG Table. The reconfiguration time is assumed to be 1000 for these experiments.

1.5.7.4 Results First consider the results of scheduling a ten-node random task DAG using the two execution profiles. The structure and parameters of an example ten-node random DAG are shown in Figure 1.12. Detailed results of using the four approaches for Profile A are shown in Table 1.2. The definitions of the data columns are: (1) $t(\text{map}[i-1])$: the task execution time of iteration i using the mapping chosen at the end of iteration $i - 1$, denoted by $\text{map}[i-1]$; (2) $t'(\text{tab}[i-1])$: the task execution time of the mapping stored

(a)



(b)

s_i	a_i	b_i	c_i
s_0	9	24	49
s_1	42	61	9
s_2	42	50	43
s_3	2	34	47
s_4	9	10	38
s_5	33	59	76
s_6	63	45	29
s_7	56	14	54
s_8	48	68	10
s_9	36	25	29

(c)

	d_{ij}	e_{ij}
$s_0 \to s_1$	5	3
$s_0 \to s_2$	6	6
$s_0 \to s_3$	2	1
$s_0 \to s_4$	7	5
$s_0 \to s_5$	3	7
$s_0 \to s_8$	5	6
$s_1 \to s_7$	3	2
$s_2 \to s_9$	7	7
$s_4 \to s_6$	4	2
$s_6 \to s_7$	5	6
$s_6 \to s_8$	7	8
$s_6 \to s_9$	9	10

(d)

	h_{i0}	h_{i1}	h_{i2}	h_{i3}
s_0	0.4897	0.6815	0.7711	0.7503
s_1	0.2828	0.7129	0.4511	0.2725
s_2	0.2575	0.8511	0.5096	0.8779
s_3	0.6337	0.6921	0.8479	0.3451
s_4	0.8283	0.2745	0.4114	0.2836
s_5	0.7267	0.3124	0.2600	0.3354
s_6	0.3932	0.7026	0.8072	0.8066
s_7	0.5276	0.7990	0.5081	0.7942
s_8	0.5876	0.3863	0.6515	0.6472
s_9	0.8760	0.8794	0.6965	0.2407

Fig. 1.12 (a) An example of a ten-node randomly generated task graph; (b) coefficients of the subtask execution time equations; (c) coefficients of the inter-subtask communication data equations; (d) heterogeneity factors h_{iu} for the subtask execution time equation.

in the MDDG Table, denoted by tab$[i-1]$, of the sample vector whose region includes the dynamic parameter values at iteration $i-1$; (3) rc: the reconfiguration time, if remapping is performed; (4) $t(\text{ect}[i-1])$: the execution time of the mapping, denoted by ect$[i-1]$, determined using the ECT algorithm with the parameters at iteration $i-1$; (5) $t(\text{ga}[i-1])$: the task execution time of iteration i by applying the mapping determined by the GA using the dynamic parameter values from iteration $i-1$; and (6) $t(\text{ga}[i])$: the task execution time of iteration i determined by the GA using the exact dynamic parameter values for iteration i.

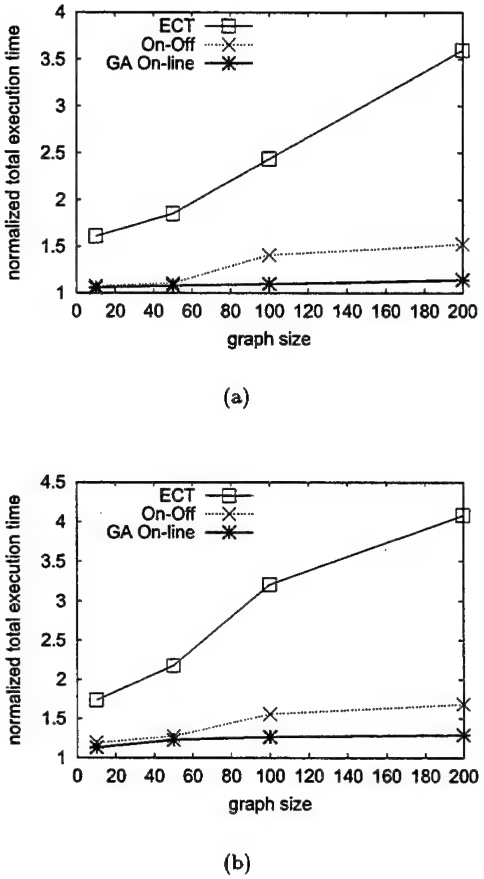
As can be seen from Table 1.2, the On-Off approach of dynamically using off-line derived mappings resulted in much smaller total execution time (1,115,545) compared to that of using the ECT algorithm (1,668,705). The improvement is approximately 33%. The On-Off approach consistently resulted in performance that was comparable to the infeasible GA On-line scheme (about 2% worse) and was only marginally outperformed by the Ideal (but

	On-Off			ECT			GA On-line	Ideal
i	t(map[i–1])	t'(tab[i–1])	rc	t(map[i–1])	t(ect[i–1])	rc	t(ga[i–1])	t(ga[i])
0	--	54391	1000	--	74516	1000	--	46228
1	47508	46045	1000	76384	76029	0	47508	47508
2	47993	47286	0	79227	75866	1000	49095	45280
3	48301	51679	0	75375	74586	0	50306	45856
4	48921	51679	0	76912	74755	1000	48520	43611
5	44035	51679	0	74966	71491	1000	53579	40937
6	49121	51679	0	76226	75020	1000	48410	46496
7	53225	52891	0	79468	77336	1000	51161	47580
8	55303	52891	1000	81944	79878	1000	55447	52809
9	56355	51679	1000	83053	82556	0	52846	50539
10	63288	63958	0	86042	87381	0	55887	53733
11	56631	52610	1000	82874	88226	0	58984	51973
12	58025	55142	1000	87122	81860	1000	63183	53733
13	57410	63958	0	87644	87182	0	57948	54311
14	59774	53073	1000	85449	87898	0	56267	52328
15	58932	60754	0	86152	87311	0	59472	55476
16	55878	58723	0	81710	81883	0	55878	55878
17	58200	59774	0	81434	76982	1000	56981	51227
18	60966	63958	0	88524	81581	1000	56961	54483
19	63071	63958	0	91701	84474	1000	60043	59846
20	65608	--	--	95498	93014	--	58528	53842
total	1,115,545			1,668,705			1,097,004	1,065,040

Table 1.2 Results for the ten-node random graph using Profile *A* ("total" below is the total task execution time for 20 iterations).

impossible) method (about 5% worse). Indeed, one very interesting observation is that at some iterations (i.e., iterations 2, 3, 5, 8, 11, 12, 13, and 15), the On-Off approach generated a mapping resulting in a shorter execution time than the GA On-line approach. Thus, the GA On-line approach, because it computes a mapping optimized for the specific dynamic parameter values from the previous iteration, is sometimes not as robust to changes in the dynamic parameter values as the region-sampling techniques used by the On-Off approach.

Figures 1.13(a) and (b) show the average normalized total execution times of the ECT, On-Off, and GA On-line approaches with respect to the Ideal method for the randomly structured graphs. The normalized total execution time of each test case is calculated by dividing the total execution time of a particular approach (e.g., ECT) by that of the Ideal method. Each point on the curves gives the average value of ten test cases. As can be seen, the normalized total execution times are, in general, slightly higher for Profile *B* than for Profile *A*. The normalized total execution times of On-Off and



(a)

(b)

Fig. 1.13 Comparison of normalized total task execution times for randomly structured graphs with (a) Profile A and (b) Profile B.

GA On-line are consistently of similar values for all graph sizes. However, the ECT approach performed much worse, especially for large graphs (sizes 100 and 200). An explanation for this phenomenon is that because the ECT algorithm employs a strictly greedy scheduling method, the effect of making mistakes at early stages of scheduling can be propagated to later stages. The adverse impact of such a greedy approach can be more profound for larger graphs. For more detailed results, the reader is referred to [34, 35].

1.5.8 Summary

This study focused on a design of a semi-static approach for using genetic algorithm derived mappings in real time. For the computational environment considered, an HC Kernel was presented for making real-time, on-line, input-data dependent remappings of the application subtasks to the processors in the heterogeneous parallel hardware platform using previously stored off-line, statically determined mappings. In particular, it was shown how the HC Kernel can be used to create the MDDG Table off-line using a genetic algorithm with a novel dynamic parameter space partitioning and sampling technique, and then use it to make real-time, on-line decisions and selections of mappings. The simulation results indicated that the semi-static On-Off approach is effective in that it consistently outperformed a fast dynamic mapping heuristic by a considerable margin, and gave reasonable performance even when compared to the impossible approach of using the genetic algorithm on-line with future knowledge of the next iteration's dynamic parameters. The On-Off approach reviewed here can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the applications and platforms considered here.

1.6 STATIC MATCHING AND SCHEDULING FOR META-TASKS

1.6.1 Introduction

This study implemented eleven different static meta-task mapping heuristics, including two evolutionary approaches, so a comparison could be made of their performance using a common simulated HC environment [6]. Subsection 1.6.2 presents information about how the ETC matrices were generated. Descriptions of the eleven heuristics implemented appear in Subsection 1.6.3. Lastly, a sampling of results from the experiments are shown in Subsection 1.6.4.

1.6.2 ETC Matrices

For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible HC environments. The ETC matrices used were generated using the following method. Initially, a $|T| \times 1$ baseline column vector, $\underline{B}$, of floating point values is created. Let $\underline{\phi_b}$ be the upper-bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $\underline{x_b^i} \in [1, \phi_b)$, and letting $B(i) = x_b^i$ for $0 \leq i < |T|$. Next, the rows of the ETC matrix are constructed. Each element ETC(i, j) in row i of the ETC matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper-bound of ϕ_r. This new random number, $x_r^{i,j} \in [1, \phi_r)$, is called a row multiplier. One row requires $|M|$ different row multipliers, $0 \leq j < |M|$. Each row i of the ETC matrix can be then described as ETC(i,j)= $B(i) \times x_r^{i,j}$, for $0 \leq j < |M|$. (The baseline column itself does not appear in the final ETC matrix.) This process is repeated for each row until the $|M| \times |T|$ ETC matrix is full. Therefore, any given value in the ETC matrix is within the range $[1, \phi_b \times \phi_r)$.

To evaluate the heuristics for different mapping scenarios, the characteristics of the ETC matrix were varied based on several different methods from [3]. The amount of variance among the execution times of tasks in the meta-task for a given machine is defined as task heterogeneity. Task heterogeneity was varied by changing the upper-bound of the random numbers within the baseline column vector. High task heterogeneity was represented by $\phi_b = 3000$ and low task heterogeneity by $\phi_b = 100$. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the upper-bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using $\phi_r = 1000$, while low machine heterogeneity values used $\phi_r = 10$. These heterogeneous ranges are based on one type of expected environment for MSHN. The ranges were chosen to reflect the fact that in real situations there is more variability across task execution times on a given machine than the execution time for a single task across different machines.

To further vary the ETC matrix in an attempt to capture more aspects of realistic mapping situations, different ETC matrix consistencies were used. An ETC matrix is said to be consistent if whenever a machine j executes any task i faster than machine k, then machine j executes all tasks faster than machine k [3]. Consistent matrices were generated by sorting each row of the ETC matrix independently. In contrast, inconsistent matrices characterize the situation where machine j is faster than machine k for some tasks, and slower for others. These matrices are left in the unordered, random state in which they were generated. Semi-consistent matrices are inconsistent matrices that include a consistent submatrix. For the semi-consistent matrices used here, the row elements in column positions $\{0, 2, 4, \ldots\}$ of row i are extracted, sorted, and replaced in order, while the row elements in column positions $\{1, 3, 5, \ldots\}$ remain unordered. (That is, the even columns are consistent and the odd columns are, in general, inconsistent.)

Sample ETC matrices are shown in Tables 1.3 and 1.4. All results in this study used ETC matrices that were of size $|T| = 512$ tasks by $|M| = 16$ machines. While it was necessary to select some specific parameter values to allow implementation of a simulation, the characteristics and techniques presented here are completely general. Therefore, if these parameter values do not apply to a specific situation of interest, researchers may use other ranges, distributions, matrix sizes, etc.

	machines							
	436,735	815,309	891,469	1,722,197	1,340,988	740,028	1,749,673	251,140
t	950,470	933,830	2,156,144	2,202,018	2,286,210	2,779,669	220,536	1,769,184
a	453,126	479,091	150,324	386,338	401,682	218,826	242,699	11,392
s	1,289,078	1,400,308	2,378,363	2,458,087	351,387	925,070	2,097,914	1,206,158
k	646,129	576,144	1,475,908	424,448	576,238	223,453	256,804	88,737
s	1,061,682	43,439	1,355,855	1,736,937	1,624,942	2,070,705	1,977,650	1,066,470
	10,783	7,453	3,454	23,720	29,817	1,143	44,249	5,039
	1,940,704	1,682,338	1,978,545	788,342	1,192,052	1,022,914	701,336	1,052,728

Table 1.3 Sample 8×8 excerpt from an ETC matrix with inconsistent, high task, high machine heterogeneity.

	machines							
	512	268	924	494	611	606	921	209
t	8	16	23	19	27	22	19	8
a	228	238	107	180	334	88	192	125
s	345	642	136	206	559	349	640	664
k	117	235	149	71	136	363	182	359
s	240	412	259	319	237	338	178	537
	462	93	574	449	421	559	487	298
	119	36	224	194	176	156	182	192

Table 1.4 Sample 8×8 excerpt from an ETC matrix with inconsistent, low task, low machine heterogeneity.

1.6.3 Description of Heuristics

The definitions of the eleven static meta-task mapping heuristics are provided below. First, some preliminary terms must be defined. Machine availability time, avail(j), is the earliest time a machine j can complete the execution of all the tasks that previously have been assigned to it. The completion time for a new task i on machine j is ct(i, j), is the machine availability time plus the execution time of task i on machine j, i.e., ct(i, j) = avail(j) + ETC(i, j). The performance criterion used to compare the results of the heuristics is the maximum value of ct(i, j), for $0 \leq i < |T|$ and $0 \leq j < |M|$, for each heuristic, also known as the makespan [39]. Each heuristic is attempting to minimize the makespan (i.e., finish execution of the meta-task as soon as possible).

The descriptions below implicitly assume that the machine availability times are updated after each task is mapped. For cases when tasks can be considered in an arbitrary order, the order in which the tasks appeared in the ETC matrix was used. Some of the heuristics listed below had to be modified from their original implementation to better handle the environment under consideration.

For many of the heuristics, there are control parameter values and/or control function specifications that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and/or information in the literature. A more thorough description of each of the heuristics and some of the intuition behind them, along with a listing of some alternative implementations, can be found in [6].

OLB: Opportunistic Load Balancing (OLB) assigns each task, in arbitrary order, to the next available machine, regardless of the task's expected execution time on that machine [2, 21, 22].

UDA: In contrast to OLB, User-Directed Assignment (UDA) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine's availability. UDA is sometimes referred to as Limited Best Assignment (LBA), as in [2, 21]. In general, this heuristic is obviously not applicable to HC environments characterized by consistent ETC matrices.

Fast Greedy: Fast Greedy assigns each task, in arbitrary order, to the machine with the minimum completion time for that task [2].

Min-min: The Min-min heuristic begins with the set U of all unmapped tasks. Then, the set of minimum completion times, MCT = $\{$ mct$_i$: mct$_i$ = min$_{0 \leq j < |M|}$(ct(i, j)), for each $i \in U$ $\}$, is found. Next, the task from U with the overall *minimum* completion time is selected and assigned to the corresponding machine (hence the name Min-min). Lastly, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [2, 21, 29].

Max-min: The Max-min heuristic is very similar to Min-min. The Max-min heuristic also begins with the set U of all unmapped tasks. Then, the set of minimum completion times, MCT is found. Next, the task from U with the overall *maximum* completion time is selected and assigned to the corresponding machine (hence the name Max-min). Lastly, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [2, 21, 29].

Greedy: The Greedy heuristic is literally a combination of the Min-min and Max-min heuristics. The Greedy heuristic performs both of the Min-min and Max-min heuristics, and uses the better solution [2, 21].

GA: The Genetic Algorithm implemented in this study was adapted from [58] (see Section 1.4 for a description of [58]) to be applied to meta-tasks, and unless otherwise noted, uses similar values and techniques. The GA operates on a population of 200 chromosomes for a given meta-task. Each chromosome is a $|T| \times 1$ vector, where position i ($0 \leq i < |T|$) represents task i, and the entry in position i is the machine to which the task has been mapped. The makespan is the fitness value. The initial population is generated using two methods: (a) 200 randomly generated chromosomes from a uniform distribution, or (b) one chromosome (seed) that is the Min-min solution and 199 random solutions (mappings). The probability of crossover was 60% and mutation was 40%. The stopping criteria that usually occurred in testing was no change in the elite chromosome in 150 iterations. Eight GA runs were performed (four times with different initial populations from each method), and the best of the eight mappings is used as the final solution.

SA: Simulated Annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This technique uses the same representation for a solution as the chromosome for the GA. Mutations of the current chromosome are also performed similarly to the GA.

The corresponding selection process for SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [13, 33, 42]). This probability is based on a system temperature that decreases for each iteration. As the system temperature "cools," it is more difficult for currently poorer solutions to be accepted. The initial system temperature is the makespan of the initial (random) mapping.

Therefore, the SA begins with the current chromosome, mutates it, and then uses the system temperature to determine whether to accept or reject this new solution. After each mutation, the system temperature is decreased by 10%. This represents one iteration of SA. The heuristic stops when there is no change in the makespan of the solution for 150 iterations or the system temperature reaches zero.

GSA: The Genetic Simulated Annealing (GSA) heuristic is a combination of the GA and SA techniques [10, 45]. In general, GSA follows procedures similar to the GA outlined above. GSA operates on a population of 200 chromosomes, uses a Min-min seed in four out of eight initial populations, and performs similar mutation and crossover operations. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simplified SA decision process for accepting or rejecting new chromosomes. GSA also employs elitism to guarantee that the best solution always remains in the population.

Tabu: Tabu search is a solution space search that keeps track of the regions of the solution space that have already been searched so as not to repeat a search near these areas [15, 24]. A solution (mapping) uses the same representation as a chromosome in the GA approach.

The implementation of Tabu search used here begins with a random mapping, generated from a uniform distribution. Starting with the first task in the mapping, task $i = 0$, each possible pair of tasks is formed, (i, j) for $0 \leq i < |T| - 1$ and $i < j < |T|$. As each pair of tasks is formed, they potentially exchange machine assignments. This constitutes a short hop. After each exchange, the new makespan is evaluated. If the new makespan is an improvement, the new exchange is retained, forming a new mapping (a successful short hop). New short hops are generated until a maximum number of successful hops have been made (see next paragraph) or all combinations of task pairs have been exhausted with no further improvement.

At this point, the final mapping from the local solution space search is added to the tabu list. Next, a new random mapping is generated, and it must differ from each mapping in the tabu list by at least half of the machine assignments (a successful long hop). Then the short hops are repeated. The final stopping criterion for the heuristic is a total of 1200 successful hops (short and long combined). Then, the best mapping from the tabu list is the final answer.

A*: A* has been applied to many other task allocation problems (e.g., [11, 31, 42, 44]). The technique used here is similar to [11].

A* is a tree search beginning at a root node that is a null solution. As the tree grows, intermediate nodes represent partial solutions (a subset of tasks are assigned to machines). The partial solution of a child node has one more task mapped than the parent node. Call this additional task $\underline{a}$. Each parent node generates $|M|$ children, one for each possible mapping of a. Based on experimentation and a desire to keep execution time of the heuristic tractable, the maximum number of leaf nodes in the tree at any one time is limited in this study to $\underline{n_{max}} = 1024$.

Each node, $\underline{n}$, has a cost function, $\underline{f(n)}$, associated with it. The cost function is an estimated lower bound on the makespan of the best solution that includes the partial solution represented by node n. (The lower bound on the time for executing the remaining tasks includes the assumptions that each task is assigned to its preferred machine and that all machines are equally utilized.)

Thus, beginning with the root, the node with the minimum $f(n)$ is expanded by its $|M|$ children, until n_{max} leaf nodes are created. From that point on, any time a node is added, the tree is pruned by deleting the leaf node with the largest $f(n)$. This process continues until a leaf node representing a complete mapping is reached. Note that if the tree is not pruned, this method is equivalent to an exhaustive search.

1.6.4 Results

An interactive software tool has been developed that allows simulation, testing, and demonstration of the heuristics examined in Subsection 1.6.3 applied to the meta-tasks defined by the ETC matrices described in Subsection 1.6.2. The software allows a user to specify $|T|$ and $|M|$, to select which types of ETC matrices to use, and to choose which heuristics to execute. It then generates the specified ETC matrices, executes the desired heuristics, and displays the results, similar to Figure 1.14. The results discussed in this section were generated using portions of this software.

When comparing mapping heuristics, the execution time of the heuristics themselves is an important consideration. For the heuristics listed, the execution times varied greatly. The experimental results discussed below were obtained on a Pentium II 400 MHz processor with 1GB of RAM. Each of the simpler heuristics (OLB, UDA, Fast Greedy, and Greedy (which includes both Min-min and Max-min)) executed in a few seconds for one ETC matrix with $|T| = 512$ and $|M| = 16$. For the same sized ETC matrix, SA and Tabu, both of which manipulate a single solution during an iteration, averaged less than 30 seconds. GA and GSA required approximately 60 seconds per matrix be-

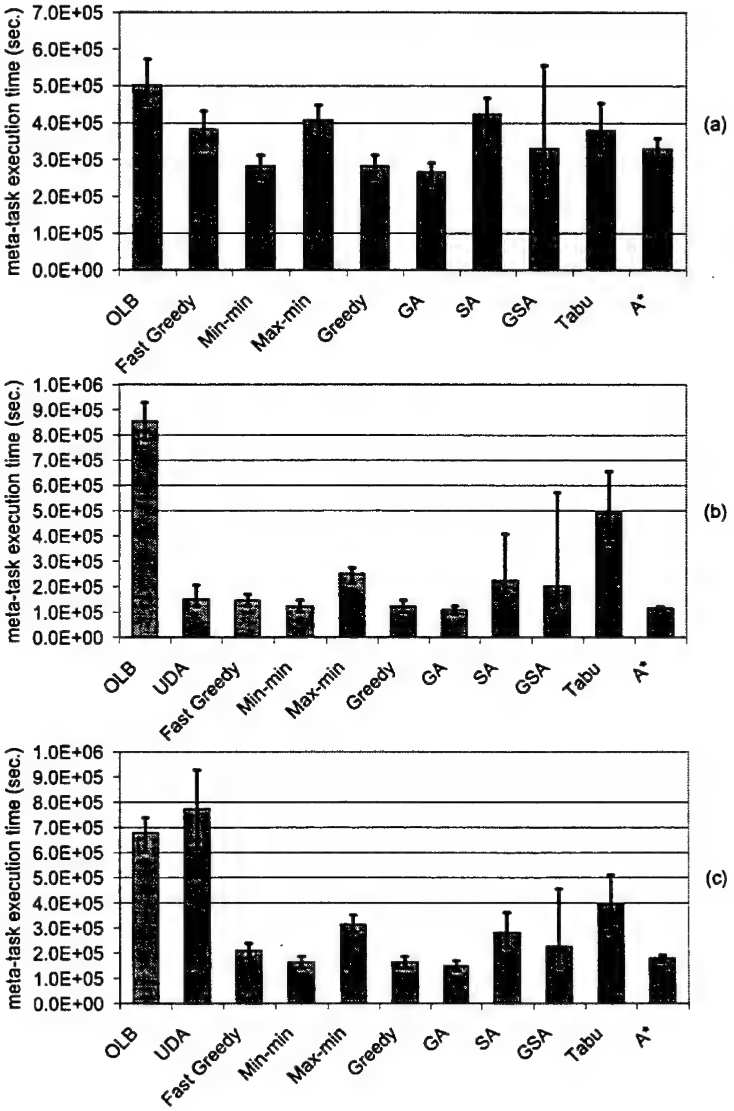


Fig. 1.14 Meta-task mapping results for 512 low-heterogeneity tasks and 16 high-heterogeneity machines for ETCs that are (a) consistent, (b) inconsistent, and (c) semi-consistent. The graphs show the mean and range over 100 trials.

cause they manipulate entire populations, and A* required about 20 minutes per matrix.

The resulting meta-task execution times (makespans) from the simulations of sample HC environments are shown in Figure 1.14. All experimental results represent the execution time of a meta-task (defined by a particular ETC matrix) based on the mapping found by the heuristic specified, averaged over 100 different ETC matrices of the same type (i.e., 100 mappings). For each heuristic, the range bars show the minimum and maximum meta-task execution times over the 100 mappings (100 ETC matrices) used to compute the average meta-task execution time.

For the four consistent cases (i.e., each combination of high and low task and machine heterogeneity), the UDA algorithm mapped all tasks to the same machine, resulting in the worst performance by an order of magnitude (therefore, UDA is not included in Figure 1.14(a)). GA performed the best for the consistent cases. This was due in large part to the good performance of the Min-min heuristic. The best GA solution always came from one of the populations that had been seeded with the Min-min solution. As is apparent in the figure, Min-min performed very well on its own, giving the second best results. However, the mutation, crossover, and selection operations of the GA were always able to improve on this solution. GSA, which also used a Min-min seed, did not always improve upon the Min-min solution. Because of the probabilistic procedure used during selection, GSA would sometimes accept poorer intermediate solutions. These poorer intermediate solutions never led to better final solutions, thus GSA gave poorer results than the GA. The performance of A* was hindered because the estimates made by $f(n)$ are not as accurate for consistent cases as they are for inconsistent and semi-consistent cases.

These results suggest that if the best overall solution is desired, the GA should be employed. However, the improvement of the GA solution over the Min-min solution was never more than 10%. Therefore, the Min-min heuristic may be more appropriate in certain situations, given the difference in execution times of the two heuristics.

For the four inconsistent test cases, UDA performs very well while the performance of OLB degrades. The OLB performance degradation for the inconsistent cases can likely be attributed to more "unfavorable" assignments occurring as compared to the consistent cases. For example, for the consistent cases, one machine executes all of the tasks quickest, so more tasks will be assigned to this machine, which is again the best assignment for that task. This phenomena is less likely to occur for the inconsistent cases because there is no one "best" machine for all of the tasks. In contrast, UDA improves because the "best" machines are distributed across the set of machines, thus task assignments will be more evenly distributed among the set of machines avoiding load imbalance (to some extent, this is due to the random distributions used in the simulation). Similarly, Fast Greedy and Min-min performed very well, and slightly outperformed UDA, because the machines providing the best task completion times are more evenly distributed among the set of machines. Min-min was also better than Max-min for all of the inconsistent cases. The advantages Min-min gains by mapping "best-case" tasks first outweighs the possible savings in "packing" that Max-min has by mapping "worst-case" tasks first [6].

Tabu gave the second poorest results for the inconsistent cases, with makespans that were always at least 20% worse than Max-min (the third poorest heuristic in terms of mean performance). Inconsistent matrices generated more successful short hops than the associated consistent matrices. Therefore, fewer long hops were generated and less of the solution space was searched, resulting in poorer solutions.

GA and A* had the best average makespans, and were usually within a small constant factor of each other. GA again benefited from having the Min-min initial mapping. A* did well because if the tasks get more evenly distributed among the machines, this more closely matches the lower-bound estimates of $f(n)$.

For semi-consistent cases with high machine heterogeneity, the UDA heuristic again gave the worst results. Intuitively, UDA is suffering from the same problem as in the consistent cases: at least half of all tasks are getting assigned to the same machine. OLB does poorly for high machine heterogeneity cases because worst-case matchings will have higher execution times for high machine heterogeneity. For low machine heterogeneity, the worst-case matchings have a much lower penalty. The best heuristics for the semi-consistent cases were Min-min and GA. This is not surprising because these were two of the best heuristics from the consistent and inconsistent tests, and semi-consistent matrices are a combination of consistent and inconsistent matrices. Min-min was able to do well because it searched the entire row for each task and assigned a high percentage of tasks to their first choice machine. GA was robust enough to handle the consistent components of the matrices, and did well for the same reason mentioned for inconsistent matrices.

1.6.5 Summary

The goal of this study was to provide insights and a basis for comparison of eleven different heuristics for the mapping of static meta-tasks in different HC environments. The characteristics of the ETC matrices used as input for the heuristics and the methods used to generate them were specified. The implementation of a collection of eleven heuristics from the literature was described. The results of the mapping heuristics were discussed, revealing the best heuristics to use in certain environments. For the situations, implementations, and parameter values used here, GA was the best heuristic for most cases, followed closely by Min-min, with A* also doing well for inconsistent matrices. The comparisons in this study can be used by researchers as a baseline for evaluating the efficacy of new techniques.

1.7 SUMMARY

In a mixed-machine, distributed, heterogeneous computing environment, there is a suite of high-performance machines with different computational capabilities. These machines are interconnected by high-speed links. Such a suite of machines can be used to execute a single application, whose subtasks have diverse execution requirements, or to execute a meta-task, which is a collection of independent tasks with different computational needs.

For the single application case, subtasks are assigned to and executed on the machines that will result in a minimal execution time for the overall task, considering subtask computation time and inter-machine communication overhead. A genetic-algorithm-based approach was described for matching subtasks to machines and scheduling the execution of the subtasks. This approach is used off-line and is based on expected computation and communication times for the subtasks. A dynamic-parameter-sampling-based method for using this approach on-line in certain application domains was also presented. This on-line adaptation may be helpful when computation and communication times can vary significantly from the expected values depending on the input data being processed.

For the meta-task case, the goal is to match each task to a machine in the suite so that the execution time for the entire meta-task is minimized. In the situation considered here, the matching process occurs off-line and plans the machine assignments and execution schedule for a meta-task for a later time interval (e.g., a large set of production jobs that will execute the next day). In this situation, the matching process can also be used to determine if a proposed set of machines can perform the meta-task within some time limit. The use of a genetic-algorithm-based approach for this environment was discussed.

In summary, much work has been done using genetic algorithms of various types to solve the problem of matching and scheduling of tasks and meta-tasks in a mixed-machine, distributed, heterogeneous computing environment. This chapter has discussed three types of genetic algorithms that have been studied to solve this problem. Each implementation has particular specifications and qualifications that were being met. In all cases, the genetic algorithm proved to be a useful method for solving the matching and scheduling problem being researched.

Acknowledgments

The authors would like to thank N. Beck and A. Naik for their useful comments. We acknowledge the contributions of our coauthors of the papers [6, 8, 35, 58], which form the basis for this chapter: I. Ahmad, N. Beck, L.L. Bölöni, J.R. Budenske, R.F. Freund, A. Ghafoor, D. Hensgen, M. Maheswaran, R.S. Ramanujan, A.I. Reuther, J.P. Robertson, R. Roychowdhury, L. Wang, and B. Yao.

REFERENCES

1. H.M. Alnuweiri and V.K. Prasanna, "Parallel architectures and algorithms for image component labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 10, Oct. 1992, pp. 1014-1034.

2. R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79-87.

3. R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance*, Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA, Sep. 1997 (D. Hensgen, advisor).

4. P. Baglietto, M. Maresca, M. Migliardi, and N. Zingirian, "Image processing on high-performance RISC systems," *Proceedings of the IEEE*, Vol. 84, No. 7, July 1996, pp. 917-930.

5. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," *IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1998, pp. 330-335 (included in the proceedings of the *7th IEEE Symposium on Reliable Distributed Systems*, 1998).

6. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," *8th IEEE Workshop on Heterogeneous Computing Systems (HCW '99)*, Apr. 1999, pp. 15-29.

7. J.R. Budenske, R.S. Ramanujan, and H.J. Siegel, "Modeling ATR applications for intelligent execution upon a heterogeneous computing platform," *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 1997, pp. 649-656.

8. J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms," *The Journal of Supercomputing*, Vol. 12, No. 4, Oct. 1998, pp. 387-406.

9. E.A. Carmona and M.D. Rice, "Modeling the serial and parallel fractions of a parallel program," *Journal of Parallel and Distributed Computing*, Vol. 13, No. 3, Nov. 1991, pp. 286-298.

10. H. Chen, N. S. Flann, and D. W. Watson, "Parallel genetic simulated annealing: A massively parallel SIMD approach," *IEEE Transactions on Parallel and Distributed Computing*, Vol. 9, No. 2, Feb. 1998, pp. 126-136.

11. K. Chow and B. Liu, "On mapping signal processing algorithms to a heterogeneous multiprocessor system," *1991 International Conference on Acoustics, Speech, and Signal Processing - ICASSP 91*, Vol. 3, May 1991, pp. 1585-1588.

12. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, MA, 1992.

13. M. Coli and P. Palazzari, "Real time pipelined system design through simulated annealing," *Journal of Systems Architecture*, Vol. 42, No. 6-7, Dec. 1996, pp. 465-475.

14. L. Davis, ed., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, NY, 1991.

15. I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro, "Improving search by incorporating evolution principles in parallel tabu search," *1994 IEEE Conference on Evolutionary Computation*, Vol. 2, 1994, pp. 823-828.

16. T. H. Einstein, "Mercury Computer Systems' modular heterogeneous RACE multicomputer," *6th IEEE Heterogeneous Computing Workshop (HCW '97),* Apr. 1997, pp. 60-71.

17. M. M. Eshaghian, ed., *Heterogeneous Computing,* Artech House, Norwood, MA, 1996.

18. M. M. Eshaghian and M. E. Shaaban, "Cluster-M programming paradigm," *International Journal on High Speed Computing,* Vol. 6, No. 2, June 1994, pp. 287-309.

19. D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering,* Vol. SE-15, No. 11, Nov. 1989, pp. 1427-1436.

20. R. F. Freund, "Optimal selection theory for superconcurrency," *Supercomputing '89,* Nov. 1989, pp. 699-703.

21. R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop (HCW '98),* Mar. 1998, pp. 184-199.

22. R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer,* Vol. 26, No. 6, June 1993, pp. 13-17.

23. A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78-86.

24. F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, MA, 1997.

25. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

26. D. A. Hensgen, T. Kidd, M. C. Schnaidt, D. St. John, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J-K. Kim, C. Irvine, T. Levin, R. Wright, R. F. Freund, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An overview of MSHN: A Management System for Heterogeneous Networks," *8th IEEE Workshop on Heterogeneous Computing Systems (HCW '99)*, Apr. 1999, pp. 184-198.

27. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, MI, 1975.

28. R. Hoebelheinrich and R. Thomsen, "Multiple crossbar network integrated supercomputing framework," *Supercomputing '89*, Nov. 1989, pp. 713-720.

29. O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280-289.

30. M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *5th IEEE Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 93-100.

31. M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July-Sept. 1998, pp. 42-51.

32. A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

33. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, Vol. 220, No. 4598, May 1983, pp. 671-680.

34. Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, A. Ghafoor, and I. Ahmad, "Evaluation of a semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems," *4th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, June 1999, pp. 204-209.

35. Y.-K. Kwok, A.A. Maciejewski, H.J. Siegel, A. Ghafoor, and I. Ahmad, "Implementation and performance study of a semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems," Technical Report HKUST-CS99-15, Department of Computer Science, The Hong Kong University of Science and Technology, in preparation.

36. M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. Webster, ed., John Wiley & Sons, New York, NY, 1999, Vol. 8, pp. 679-690.

37. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing,* to appear, 1999.

38. B. Narahari, A. Youssef, and H. A. Choi, "Matching and scheduling in a generalized optimal selection theory," *3rd IEEE Heterogeneous Computing Workshop (HCW '94),* Apr. 1994, pp. 3-8.

39. M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.

40. J. L. Ribeiro Filho and P. C. Treleaven, "Genetic-algorithm programming environments," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 28-43.

41. G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks,* Vol. 5, No. 1, Jan. 1994, pp. 96-101.

42. S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.

43. K.C. Sevcik, "Characterizations of parallelism in applications and their use in scheduling," *Performance Evaluation Review,* Vol. 17, No. 1, May 1989, pp. 171-180.

44. C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing system using a minimax criterion," *IEEE Transactions on Computers*, Vol. C-34, No. 3, Mar. 1985, pp. 197-203.

45. P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th IEEE Heterogeneous Computing Workshop (HCW '96),* Apr. 1996, pp. 98-104.

46. H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and L. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook,* A. Y. Zomaya ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

47. H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook,* A. B. Tucker Jr. ed., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

48. H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th IEEE Heterogeneous Computing Workshop (HCW '96),* Apr. 1996, pp. 86-97.

49. M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer,* Vol. 27, No. 6, June 1994, pp. 17-26.

50. V. S. Sunderam, "Design issues in heterogeneous network computing," *Workshop on Heterogeneous Processing (revised edition),* Mar. 1992, pp. 101-112.

51. M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, "Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system," *IEEE Transactions on Parallel and Distributed Systems,* Vol. 8, No. 8, Aug. 1997, pp. 857-871.

52. Y. G. Tirat-Gefen, and A. C. Parker, "MEGA: An approach to system-level design of application-specific heterogeneous multiprocessors" *4th IEEE Heterogeneous Computing Workshop (HCW '96),* Apr. 1996, pp. 105-117.

53. D. Tolmie and J. Renwick, "HiPPI: Simplicity yields success," *IEEE Network,* Vol. 7, No. 1, Jan. 1993, pp. 28-32.

54. J.G. Verly and R.L Delanoy, "Model-based automatic target recognition (ATR) system for forwardlooking groundbased and airborne imaging laser radars (LADAR)," *Proceedings of the IEEE,* Vol. 84, No. 2, Feb. 1996, pp. 126-163.

55. D. W. Watson, J. K. Antonio, H. J. Siegel, and M. J. Atallah, "Static program decomposition among machines in an SIMD/SPMD heterogeneous environment with non-constant mode switching costs," *3rd IEEE Heterogeneous Computing Workshop (HCW '94),* Apr. 1994, pp. 58-65.

56. D. W. Watson, J. K. Antonio, H. J. Siegel, R. Gupta, and M. J. Atallah, "Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment," *5th IEEE Heterogeneous Computing Workshop (HCW '96),* Apr. 1996, pp. 24-37.

57. Q. Wang and K.H. Cheng, "List scheduling of parallel tasks," *Information Processing Letters,* Vol. 37, No. 5, Mar. 1991, pp. 291-297.

58. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using

a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 1-15.

59. C. C. Weems, G. E. Weaver, and S. G. Dropsho, "Linguistic support for heterogeneous parallel processing: A survey and an approach," *3rd IEEE Heterogeneous Computing Workshop (HCW '94)*, Apr. 1994, pp. 81-88.

60. D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," *1989 International Conference on Genetic Algorithms*, Morgan Kaufmann, June 1989, pp. 116-121.

61. A. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: Comparative studies and performance issues," *IEEE Transactions on Parallel and Distributed Systems*, to appear.

JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **47**, 8–22 (1997)
ARTICLE NO. PC971392

Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach[1]

Lee Wang,*,[2] Howard Jay Siegel,*,[2] Vwani P. Roychowdhury,†,[3] and Anthony A. Maciejewski*,[2]

**Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana 47907-1285;*
and †Electrical Engineering Department, UCLA, Los Angeles, California 90095-1594

To exploit a heterogeneous computing (HC) environment, an application task may be decomposed into subtasks that have data dependencies. Subtask matching and scheduling consists of assigning subtasks to machines, ordering subtask execution for each machine, and ordering intermachine data transfers. The goal is to achieve the minimal completion time for the task. A heuristic approach based on a genetic algorithm is developed to do matching and scheduling in HC environments. It is assumed that the matcher/scheduler is in control of a dedicated HC suite of machines. The characteristics of this genetic-algorithm-based approach include: separation of the matching and the scheduling representations, independence of the chromosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. It is applicable to the static scheduling of production jobs and can be readily used to collectively schedule a set of tasks that are decomposed into subtasks. Some parameters and the selection scheme of the genetic algorithm were chosen experimentally to achieve the best performance. Extensive simulation tests were conducted. For small-sized problems (e.g., a small number of subtasks and a small number of machines), exhaustive searches were used to verify that this genetic-algorithm-based approach found the optimal solutions. Simulation results for larger-sized problems showed that this genetic-algorithm-based approach outperformed two nonevolutionary heuristics and a random search. © 1997 Academic Press

1. INTRODUCTION

Different portions of an application task often require different types of computation. In general, it is impossible for a single machine architecture with its associated compiler, operating system, and programming tools to satisfy all the computational requirements in such an application equally well. However, a *heterogeneous computing* (*HC*) environment that consists of a heterogeneous suite of machines, high-speed interconnections, interfaces, operating systems, communication protocols, and programming environments provides a variety of architectural capabilities, which can be orchestrated to perform an application that has diverse execution requirements [Fre89, FrS93, KhP93, SiA96, Sun92]. In the HC environment considered here, an application task can be decomposed into subtasks, where each subtask is computationally homogeneous (well suited to a single machine), and different subtasks may have different machine architectural requirements. These subtasks can have data dependences among them. Once the application task is decomposed into subtasks, the following decisions have to be made: *matching*, i.e., assigning subtasks to machines, and *scheduling*, i.e., ordering subtask execution for each machine and ordering intermachine data transfers. In this context, the goal of HC is to achieve the minimal *completion time*, i.e., the minimal overall execution time of the application task in the machine suite.

It is well known that such a matching and scheduling problem is in general NP-complete [Fer89]. A number of approaches to different aspects of this problem have been proposed (e.g., [EsS94, Fre89, IvO95, NaY94, TaA95, WaA94]). Different from the above approaches, this paper proposes a genetic-algorithm-based approach for solving the problem.

Genetic algorithms for subtask scheduling in a collection of homogeneous processors have been considered (e.g., [AhD96, BeS94, HoA94]). Performing matching and scheduling for a suite of heterogeneous machines, however, requires a very different genetic algorithm structure.

In [IvO95], a nonevolutionary heuristic based on level scheduling [ChL88, MuC69] is presented to find a suboptimal matching and concurrent scheduling decision. That approach is compared to the performance of the evolutionary genetic-algorithm-based approach proposed in this paper.

This paper proposes a genetic-algorithm-based approach for solving the matching and concurrent scheduling problem in HC systems. It decides the subtask to machine assignments, orders the execution of the subtasks assigned to each machine, and schedules the data transfers among subtasks. The characteristics of this approach include: separation of the matching and the scheduling representations, independence of the chro-

[1] This research was supported in part by NRaD under Subcontract 20-950001-70 and by the DARPA/ITO Quorum Program under NPS Subcontract N62271-97-M-0900.

[2] E-mail: {lwang,hj,maciejew}@ecn.purdue.edu.

[3] E-mail: vwani@ee.ucla.edu.

mosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. The computation and communication overlap is limited only by intersubtask data dependencies and machine/network availability. This genetic-algorithm-based approach can be applied to performing the matching and scheduling in a variety of HC systems. It is applicable to the static scheduling of production jobs and can be readily used to collectively schedule a set of tasks that are decomposed into subtasks.

The organization of this paper is as follows. The matching and scheduling problem is defined in Section 2. Section 3 briefly describes genetic algorithms and gives the outline of the genetic-algorithm-based approach. In Section 4, the proposed representation of matching and scheduling decisions within the genetic framework is presented. Section 5 discusses how to generate the initial population of possible solutions used by the genetic algorithm. The selection mechanism is discussed in Section 6. Sections 7 and 8 define the crossover and mutation operators, respectively, used to construct new generations of populations. Section 9 gives the method for evaluating the quality of a solution and the experimental results are shown in Section 10. Some related work is viewed and compared with our approach in Section 11. Finally, Section 12 discusses some future research directions.

2. PROBLEM DEFINITION

There are many open research problems in the field of HC [SiA96]. To isolate and focus on the matching and scheduling problem, assumptions about other components of an overall HC system must be made. Assumptions such as those below are typically made by matching and scheduling researchers (e.g., [ShW96, SiY96]).

It is assumed that the application task is written in some machine-independent language (e.g., [WeW94]). It is also assumed that an application task is decomposed into multiple subtasks and the data dependencies among them are known and are represented by a directed acyclic graph. If intermachine data transfers are data dependent, then some set of expected data transfers must be assumed. The estimated expected execution time for each subtask on each machine is assumed to be known *a priori*. The assumption of the availability of expected subtask execution time for each type of machine is typically made for the current state-of-the-art in HC systems when studying the matching and scheduling problem (e.g., [Fre94, GhY93, ShW96, SiY96]). Finding the estimated expected execution times for subtasks is another research problem, which is outside the scope of this paper. Approaches for doing this estimation based on task profiling and analytical benchmarking are surveyed in [SiA96]. The HC system is assumed to have operating system support for executing each subtask on the machine it is assigned and for performing intermachine data transfers as scheduled by this genetic-algorithm-based approach.

In the type of HC system considered here, an application task is decomposed into a set of subtasks S. Define $|S|$ to be the number of subtasks in the set S and s_i to be the ith subtask. Then $S = \{s_i, 0 \le i < |S|\}$. An HC environment consists of a set of machines M. Define $|M|$ to be the number of machines in the set M and m_j to be the jth machine. Then $M = \{m_j, 0 \le j < |M|\}$. The estimated expected execution time of subtask s_i on machine m_j is T_{ij}, where $0 \le i < |S|$ and $0 \le j < |M|$. The *global data items* (*gdis*), i.e., data items that need to be transferred between subtasks, form a set G. Define $|G|$ to be the number of items in the set G and gdi_k to be the kth global data item. Then $G = \{\text{gdi}_k, 0 \le k < |G|\}$.

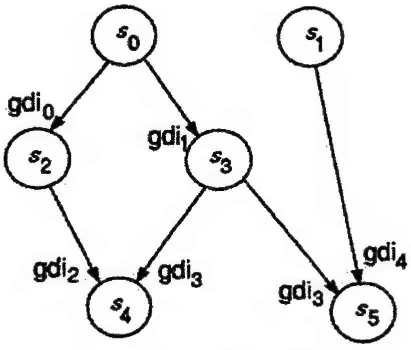


FIG. 1. An example DAG.

It is assumed that for each global data item, there is a single subtask that produces it (*producer*) and there are some subtasks that need this data item (*consumers*). The task is represented by a directed acyclic graph (*DAG*). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred. Figure 1 shows an example DAG.

The following further assumptions are made for the problem. One is the exclusive use of the HC environment for the application. The genetic-algorithm-based matcher/scheduler is in control of the HC machine suite. Another is nonpreemptive subtask execution. Also, all input data items of a subtask must be received before its execution can begin, and none of its output data items is available until the execution of this subtask is finished. If a data conditional is based on input data, it is assumed to be contained inside a subtask. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a subtask. These restrictions help make the matching and scheduling problem more manageable and solving this problem under these assumptions is a significant step forward for solving the general matching and scheduling problem.

3. GENETIC ALGORITHMS

Genetic algorithms (*GAs*) are a promising heuristic approach to finding near-optimal solutions in large search spaces [Dav91, Gol89, Hol75]. There are a great variety of approaches to GAs; many are surveyed in [SrP94, RiT94]. The following is a brief overview of GAs to provide background for the description of the proposed approach.

The first step necessary to employ a GA is to encode any possible solution to the optimization problem as a set of strings (*chromosome*). Each chromosome represents one solution to the problem, and a set of chromosomes is referred to as a *population*. The next step is to derive an initial population. A random set of chromosomes is often used as the initial population. Some specified chromosomes can also be included. This initial population is the first generation from which the evolution starts.

The third step is to *evaluate* the quality of each chromosome. Each chromosome is associated with a *fitness value*, which is in this case the completion time of the solution (matching and scheduling) represented by this chromosome (i.e., the expected execution time of the application task if the matching and scheduling specified by this chromosome were used). Thus, in this research a smaller fitness value represents a better solution. The objective of the GA search is to find a chromosome that has the optimal (smallest) fitness value. The selection process is the next step. In this step, each chromosome is eliminated or duplicated (one or more times) based on its relative quality. The population size is typically kept constant.

Selection is followed by the *crossover* step. With some probability, some pairs of chromosomes are selected from the current population and some of their corresponding components are exchanged to form two valid chromosomes, which may or may not already be in the current population. After crossover, each string in the population may be *mutated* with some probability. The mutation process transforms a chromosome into another valid one that may or may not already be in the current population. The new population is then evaluated. If the stopping criteria have not been met, the new population goes through another cycle (iteration) of selection, crossover, mutation, and evaluation. These cycles continue until one of the stopping criteria is met.

In summary, the following are the steps that are taken to implement a GA for a given optimization problem: (1) an encoding, (2) an initial population, (3) an evaluation using a particular fitness function, (4) a selection mechanism, (5) a crossover mechanism, (6) a mutation mechanism, and (7) a set of stopping criteria. These steps of a typical GA are shown in Fig. 2.

Details of the steps for the implementation of the GA-based heuristic for HC will be discussed in the following sections. For some parameters of this GA, such as population size, values were selected based on information in the literature. For other parameters, such as the probability of performing a mutation operation, experiments were conducted (Section 10).

```
GA_matching_scheduling(){
        initial population generation;
        evaluation;
        while(stopping criteria not met){
                selection;
                crossover;
                mutation;
                evaluation;
        }
        output the best solution found;
}
```

FIG. 2. The steps in a typical GA.

4. CHROMOSOME REPRESENTATION

In this GA-based approach, each chromosome consists of two parts: the matching string and the scheduling string. Let *mat* be the *matching string*, which is a vector of length $|S|$, such that mat(i) = j, where $0 \leq i < |S|$ and $0 \leq j < |M|$; i.e., subtask s_i is assigned to machine m_j.

The *scheduling string* is a topological sort [CoL92] of the DAG, i.e., a total ordering of the nodes (subtasks) in the DAG that obeys the precedence constraints. Define *ss* to be the scheduling string, which is a vector of length $|S|$, such that ss(k) = i, where $0 \leq i$, $k < |S|$, and each s_i appears only once in the vector, i.e., subtask s_i is the kth subtask in the scheduling string. Because it is a topological sort, if ss(k) is a consumer of a global data item produced by ss(j), then $j < k$. The scheduling string gives an ordering of the subtasks that is used by the evaluation step.

Then in this GA-based approach, a chromosome is represented by a two-tuple ⟨mat, ss⟩. Thus, a chromosome represents the subtask-to-machine assignments (matching) and the execution ordering of the subtasks assigned to the same machine. The scheduling of the global data item transfers and the relative ordering of subtasks assigned to different machines are determined by the evaluation step. Figure 3 illustrates two different chromosomes for the DAG in Fig. 1, for $|S| = 6$, $|M| = 3$, and $|G| = 5$.

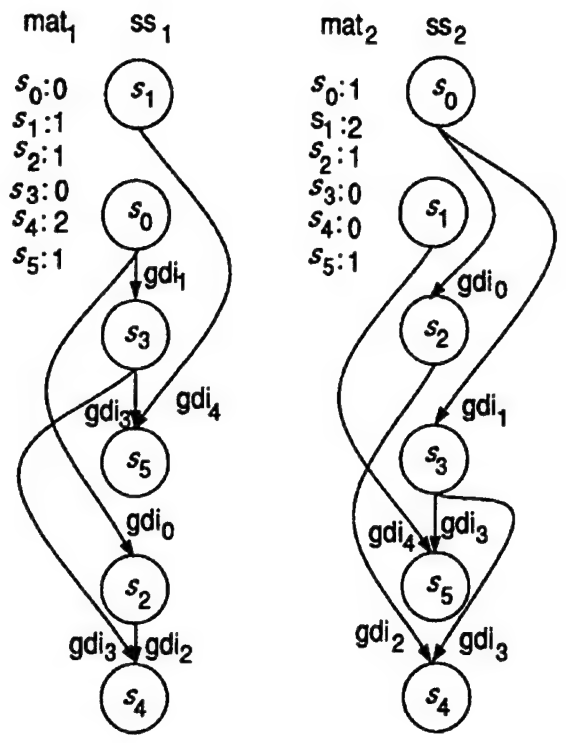


FIG. 3. Two chromosomes from the DAG in Fig. 1.

5. INITIAL POPULATION GENERATION

In the initial population generation step, a predefined number of chromosomes are generated, the collection of which form the initial population. When generating a chromosome, a new matching string is obtained by randomly assigning each subtask to a machine. To form a scheduling string, the DAG is first topologically sorted to form a basis scheduling string. Then, for each chromosome in the initial population, this basis string is mutated a random number of times (between one and the number of subtasks) using the scheduling string mutation operator (defined in Section 8) to generate the ss vector (which is a valid topological sort of the given DAG). Furthermore, it is common in GA applications to incorporate solutions from some nonevolutionary heuristics into the initial population, which may reduce the time needed for finding a satisfactory solution [Dav91]. In this GA-based approach, along with those chromosomes representing randomly generated solutions, the initial population also includes a chromosome that represents the solution from a nonevolutionary baseline heuristic. Details of this heuristic will be discussed in Section 10.

Each newly generated chromosome is checked against those previously generated. If a new chromosome is identical to any of the existing ones, it is discarded and the process of chromosome generation is repeated until a unique new chromosome is obtained. The reason why identical chromosomes are not allowed in the initial generation is that they could possibly drive the whole population to a premature *convergence*, i.e., the state where all chromosomes in a population have the same fitness value. It can be shown that for this GA-based approach, there is a nonzero probability that a chromosome can be generated to represent any possible solution to the matching and scheduling problem using the crossover and the mutation operators. The crossover and the mutation operators will be discussed later in Sections 7 and 8, respectively.

6. SELECTION

In this step, the chromosomes in the population are first ordered (ranked) by their fitness values from the best to the worst. Those having the same fitness value are ranked arbitrarily among themselves. Then a *rank-based roulette wheel selection scheme* can be used to implement the selection step [Hol75, SrP94]. In the rank-based selection scheme, each chromosome is allocated a sector on a roulette wheel. Let P denote the population size and A_i denote the angle of the sector allocated to the ith ranked chromosome. The 0th ranked chromosome is the fittest and has the sector with the largest angle A_0; whereas the $(P-1)$th ranked chromosome is the least fit and has the sector with the smallest angle A_{P-1}. The ratio of the sector angles between two adjacently ranked chromosomes is a constant $R = A_i/A_{i+1}$, where $0 \leq i < P - 1$. If the 360 degrees of the wheel are normalized to one, then

$$A_i = R^{P-i-1} \times (1 - R)/(1 - R^P),$$

where $R > 1$, $0 \leq i < P$, and $0 < A_i < 1$.

The selection step generates P random numbers, ranging from zero to one. Each number falls in a sector on the roulette wheel and a copy of the owner chromosome of this sector is included in the next generation. Because a better solution has a larger sector angle than that of a worse solution, there is a higher probability that (one or more) copies of this better solution will be included in the next generation. In this way, the population for the next generation is determined. Thus, the population size is always P, and it is possible to have multiple copies of the same chromosome.

Alternatively, a *value-based roulette wheel selection scheme* can be used to implement a proportionate selection [SrP94]. Let f_i be the fitness value of the ith chromosome and f_{ave} be the average fitness value of the current population. In this selection scheme, the ith chromosome ($0 \leq i < P$) is allocated a sector on the roulette wheel, the angle of which, A_i, is proportional to f_{ave}/f_i (assuming that the best chromosome has the smallest fitness value, which is the case for this research). The most appropriate selection scheme for this research was chosen experimentally. Details on the experiments can be found in Section 10 and [Wan97].

This GA-based approach also incorporates *elitism* [Rud94]. At the end of each iteration, the best chromosome is always compared with an elite chromosome, a copy of which is stored separately from the population. If the best chromosome is better than the elite chromosome, a copy of it becomes the elite chromosome. If the best chromosome is not as good as the elite chromosome, a copy of the elite chromosome replaces the worst chromosome in the population. Elitism is important because it guarantees that the quality of the best solutions found over generations is monotonically increasing.

7. CROSSOVER OPERATORS

Different crossover operators are developed for scheduling strings and matching strings. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings. For each pair, it randomly generates a cut-off point, which divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative positions of these subtasks in the other original scheduling string in the pair, thus guaranteeing that the newly generated scheduling strings are valid schedules. Figure 4 demonstrates such a scheduling string crossover process.

The crossover operator for the matching strings randomly chooses some pairs of the matching strings. For each pair, it randomly generates a cut-off point to divide both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The probability for performing crossovers was determined by experimentation. This is discussed in Section 10.

8. MUTATION OPERATORS

Different mutation operators are developed for scheduling strings and matching strings. The scheduling string mutation

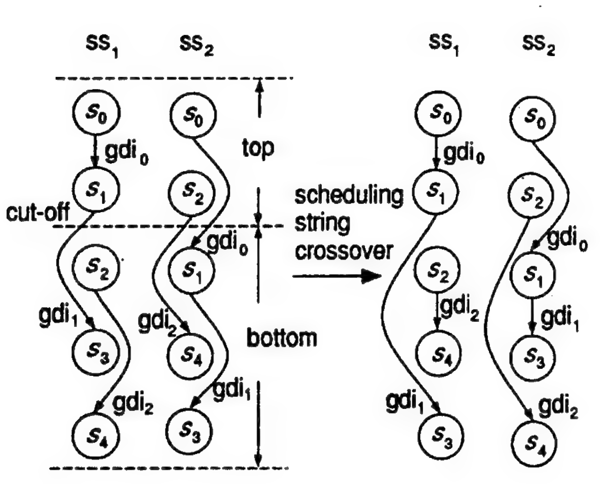


FIG. 4. A scheduling string crossover example.

operator randomly chooses some scheduling strings. Then for each chosen scheduling string, it randomly selects a victim subtask. The *valid range* of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed without violating any data dependency constraints. Specifically, the valid range is after all source subtasks of the victim subtask and before any destination subtask of the victim subtask. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its valid range. Figure 5 shows an example of this mutation process.

The matching string mutation operator randomly chooses some matching strings. On each chosen matching string, it randomly selects a subtask/machine pair. Then the machine assignment for the selected pair is changed randomly to another machine.

The probability for performing mutations was determined by experimentation. This is discussed in Section 10.

9. EVALUATION

The final step of a GA iteration is the evaluation of the fitness value of each chromosome. In this GA-based approach, the chromosome structure is independent of any particular communication subsystem. Only the evaluation step needs the communication characteristics of the given HC system to schedule the data transfers. To test the effectiveness of this GA-based approach, an example communication system was chosen. This GA-based approach can be used with any communication system that obeys the assumptions in Section 2.

To demonstrate the evaluation process, an example communication subsystem, which is modeled after a HiPPI LAN with a central crossbar switch [HoT89, ToR93], is assumed to connect a suite of machines. Each machine in the HC suite has one input data link and one output data link. All these links are connected to a central crossbar switch. Figure 6 shows an HC system consisting of four machines that are interconnected by such a crossbar switch. If a subtask needs a global data item that is produced or consumed earlier by a different subtask on the same machine, the communication time for this item is zero. Otherwise, the communication time is obtained by dividing the size of the global data item by the smaller bandwidth of the output link of the source machine and the input link of the destination machine. In this research, it is assumed that for a given machine, the bandwidths of the input link and the output link are equal to each other. It is also assumed that the crossbar switch has a higher bandwidth than that of each link. The communication latency between any pair of machines is assumed to be the same. Data transfers are neither preemptive nor multiplexed. Once a data transfer path is established, it cannot be relinquished until the data item (e.g., gdi$_k$) scheduled to be transferred over this path is received by the destination machine. Multiple data transfers over the same path have to be serialized.

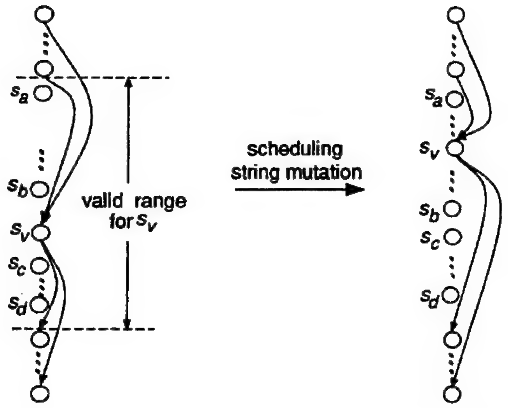


FIG. 5. A scheduling string mutation example. Only edges to and from the victim subtask s_v are shown. Before the mutation, s_v is between s_b and s_c. After the mutation, it is moved to between s_a and s_b.

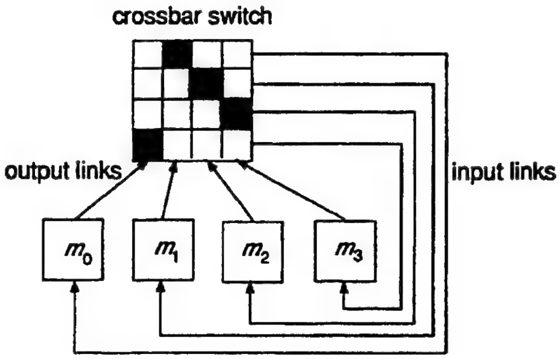


FIG. 6. An example HC system with four machines and a central crossbar switched network. Each machine has one output data link to and one input data link from the crossbar switch. Blackened squares in the switch correspond to active connections.

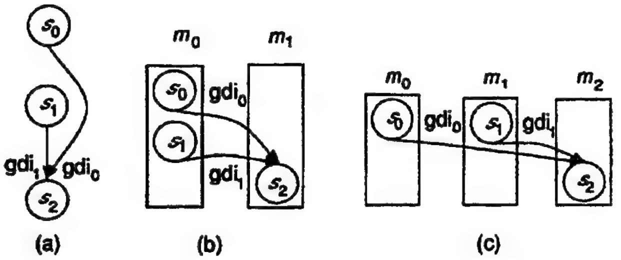


FIG. 7. An example showing the scheduling order for the input gdis of one subtask: (a) the example scheduling string; (b) the situation when the source subtasks of the input gdis are assigned to the same machine; (c) the situation when the source subtasks of the input gdis are assigned to different machines.

In this step, for each chromosome the final order of execution of the subtasks and the intermachine data transfers are determined. The evaluation procedure considers the subtasks in the order they appear on the scheduling string. Subtasks assigned to the same machine are executed exactly in the order specified by the scheduling string. For subtasks assigned to different machines, the actual execution order may deviate from that specified by the scheduling string due to factors such as input-data availability and machine availability. This is explained below.

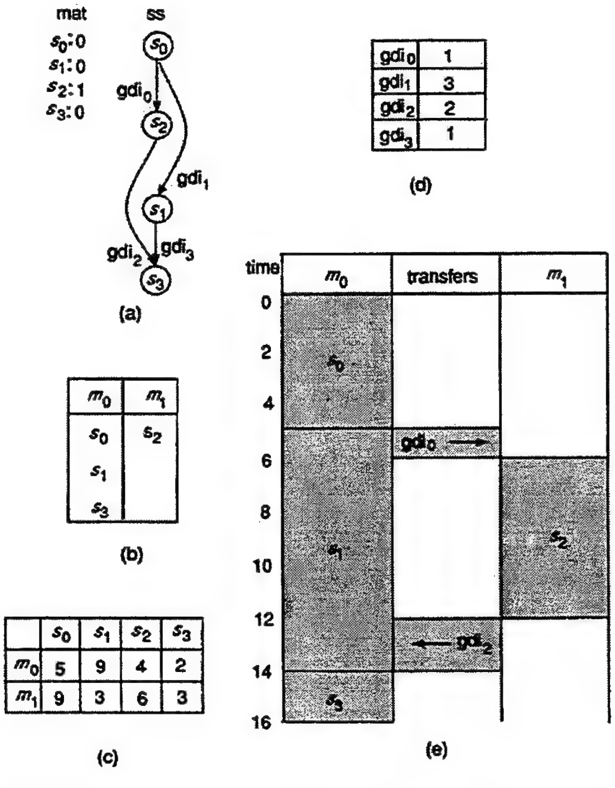
Before a subtask can be scheduled, all of its input global data items must be received. For each subtask, its input data items are considered by the evaluation procedure in the order of their producers' relative positions in the scheduling string. The reason for this ordering is to better utilize the overlap of subtask executions and intermachine data communications. The following example illustrates this idea. Let ss(0) = 0, ss(1) = 1, and ss(2) = 2, as shown in Fig. 7a. Let s_2 need two gdis, gdi$_0$ and gdi$_1$, from s_0 and s_1, respectively. Depending on the subtask to machine assignments, the data transfers of gdi$_0$ and gdi$_1$ could be either local within a machine or across machines. If at least one data transfer is local, then the scheduling is trivial because it is assumed that local transfers within a machine take negligible time. However, there exist two situations where both data transfers are across machines so that they need to be ordered.

Situation 1. Let s_0 and s_1 be assigned to the same machine m_0 and s_2 be assigned to another machine m_1, as shown in Fig. 7b. In this situation, because s_0 is to be executed before s_1, gdi$_0$ is available before gdi$_1$ becomes available on machine m_0. Thus, it is better to schedule the gdi$_0$ transfer before the gdi$_1$ transfer.

Situation 2. Let the three subtasks s_0, s_1, and s_2 be assigned to three different machines m_0, m_1, and m_2, as shown in Fig. 7c. In this situation, if there is a data dependency from s_0 to s_1, then s_0 finishes its execution before s_1 could start its execution. Therefore, gdi$_0$ is available before gdi$_1$ becomes available. Hence, it is better to schedule the gdi$_0$ transfer before the gdi$_1$ transfer. If there are no data dependencies from s_0 to s_1, the gdi$_0$ transfer will still be scheduled before the gdi$_1$ transfer. While this may not be the best scheduling order for these gdis, the reverse order may be considered by other scheduling strings, i.e., there may be some other chromosome(s) that have ss(0) = 1 and ss(1) = 0. When such a chromosome is evaluated, the gdi$_1$ transfer will be scheduled before the gdi$_0$ transfer. Therefore, it is possible for all input gdi scheduling orderings for gdi$_0$ and gdi$_1$ to be examined.

In Fig. 8, a simple example is shown to illustrate the evaluation for a given chromosome. In this example (as well as some others given later), because there are only two machines, the source and destination machines for the gdi transfers are implicit. The ordering for the evaluation of subtasks and gdi transfers is: s_0, gdi$_0$, s_2, gdi$_1$, s_1, gdi$_2$, gdi$_3$, s_3. If a gdi consumer subtask is on the same machine as the producer or as a previous consumer of that gdi, no data transfer is required, as is the case for gdi$_1$ and gdi$_3$ in this example.

Data forwarding is another important feature of this evaluation process. For each input data item to be considered, the evaluation process chooses the source subtask from among the producer of this data item and all the consumers that have received this data item. These consumers are *forwarders*. The



m_0	m_1
s_0	s_2
s_1	
s_3	

(b)

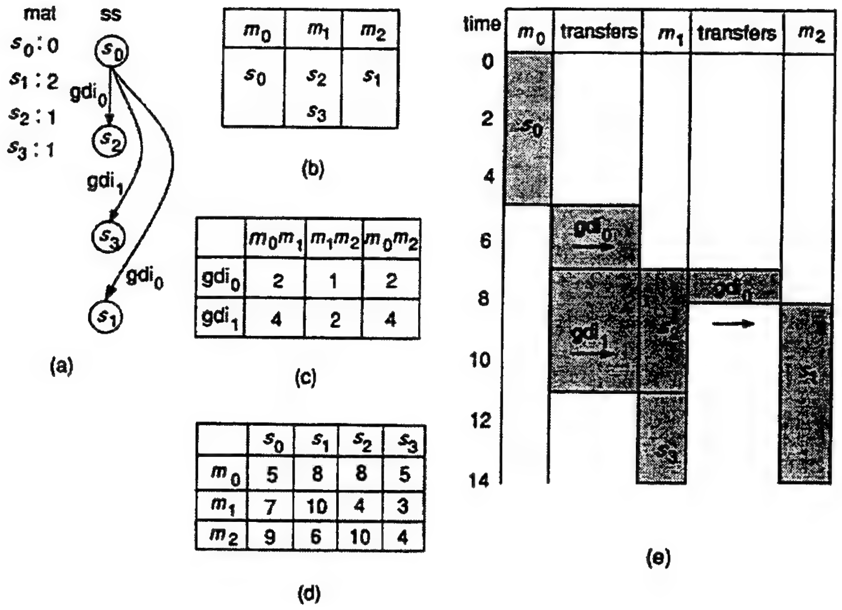
	s_0	s_1	s_2	s_3
m_0	5	9	4	2
m_1	9	3	6	3

(c)

gdi$_0$	1
gdi$_1$	3
gdi$_2$	2
gdi$_3$	1

(d)

FIG. 8. An example showing the evaluation step: (a) the chromosome; (b) the subtask execution ordering on each machine given by (a); (c) the estimated subtask execution times; (d) the gdi intermachine transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution and data transfer timings, where the completion time for this chromosome is 16.



(a)

m_0	m_1	m_2
s_0	s_2 s_3	s_1

(b)

	m_0m_1	m_1m_2	m_0m_2
gdi_0	2	1	2
gdi_1	4	2	4

(c)

	s_0	s_1	s_2	s_3
m_0	5	8	8	5
m_1	7	10	4	3
m_2	9	6	10	4

(d)

(e)

FIG. 9. A data forwarding example: (a) the chromosome; (b) the subtask execution ordering on each machine; (c) the gdi transfer times; (d) the estimated subtask execution times; and (e) the subtask execution and data transfer times using data forwarding.

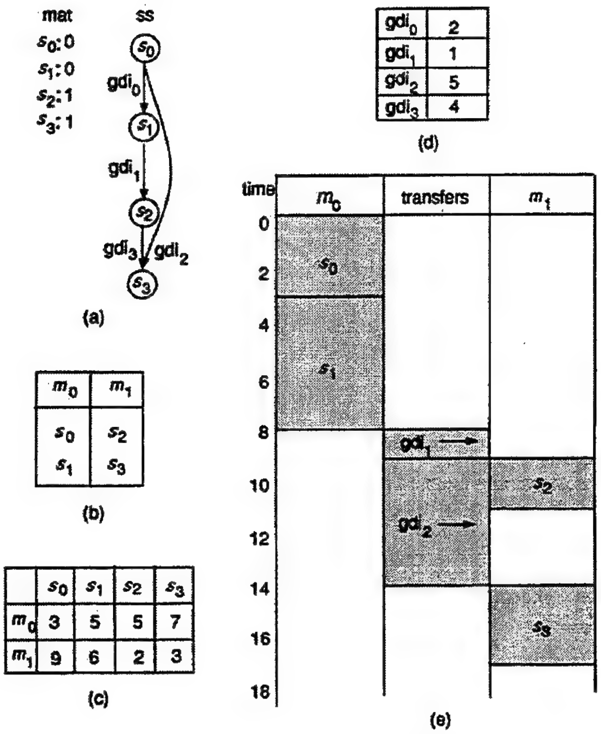
one (either the producer or a forwarder) from which the destination subtask will receive the data item at the earliest possible time is chosen. Figure 9 shows an example of data forwarding. In this example, global data item gdi_0 is forwarded to subtask s_1 from a consumer subtask s_2 instead of from the producer subtask s_0. The resulting completion time is 14. If data forwarding is disabled for this example (i.e., global data item gdi_0 must be sent from subtask s_0 to subtask s_1), the completion time would be 16 (when subtask s_0 sends gdi_0 to subtask s_1 before sending gdi_1 to subtask s_3) or 19 (when subtask s_0 sends gdi_1 to subtask s_3 before sending gdi_0 to subtask s_1).

After the source subtask is chosen, the data transfer for the input data item is scheduled. A transfer starts at the earliest point in time from when the path from the source machine to the destination machine is free for a period at least equal to the needed transfer time. This (possibly) *out-of-order* scheduling of the input item data transfers utilizes previously idle bandwidths of the communication links and thus could make some input data items available to some subtasks earlier than otherwise from the in-order scheduling. As a result, some subtasks could start their execution earlier, which would in turn decrease the overall task completion time. This is referred to as out-of-order scheduling of data transfers because the data transfers do not occur in the order in which they are considered (i.e., the *in-order* schedule). Figures 10 and 11 show the in-order scheduling and the out-of-order scheduling for the same chromosome, respectively. In the in-order scheduling, the transfer of gdi_1 is scheduled before the transfer of gdi_2 because subtask s_2's input data transfers are considered before those of subtask s_3. In this example, the out-of-order schedule does decrease the total execution time of the given task.

When two chromosomes have different matching strings, they are different solutions because the subtask-to-machine assignments are different. However, two chromosomes that have the same matching string but different scheduling strings may or may not represent the same solution. This is because the scheduling string information is used in two cases: (1) for scheduling subtasks that have been assigned to the same machine and (2) for scheduling data transfers. Two different scheduling strings could result in the same ordering for (1) and (2).

After a chromosome is evaluated, it is associated with a fitness value, which is the time when the last subtask finishes its execution. That is, the fitness value of a chromosome is then the overall execution time of the task, given the matching and scheduling decision specified by this chromosome and by the evaluation process.

In summary, this evaluation mechanism considers subtasks in the order in which they appear in the scheduling string. For a subtask that requires some gdis from other machines, the gdi transfer whose producer subtask appears earliest in the scheduling string is scheduled first. When scheduling a gdi transfer, both the producing and the forwarding subtasks are considered. The source subtask that lets this consumer subtask receive this gdi at the earliest possible time is chosen to send the gdi. The out-of-order scheduling of the gdi transfers over a path could further reduce the completion time of the application.



m_0	m_1
s_0 s_1	s_2 s_3

	s_0	s_1	s_2	s_3
m_0	3	5	5	7
m_1	9	6	2	3

gdi_0	2
gdi_1	1
gdi_2	5
gdi_3	4

FIG. 10. An example showing the in-order scheduling of a chromosome: (a) the chromosome; (b) the subtask execution ordering on each machine; (c) the estimated subtask execution times; (d) the gdi transfer times (transfers between subtasks assigned to the same machine take zero time); and (e) the subtask execution and data transfer timings using in-order transfers (the gdi_1 transfer occurs before the gdi_2 transfer), where the completion time is 17.

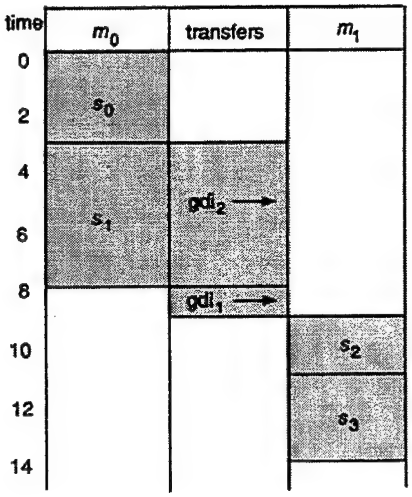


FIG. 11. An example showing the out-of-order scheduling, where the chromosome and other statistics are the same as in Fig. 10. The completion time is 14.

10. EXPERIMENTAL RESULTS

To measure the performance of this GA-based approach, randomly generated *scenarios* were used, where each scenario corresponded to a DAG, the associated subtask execution times, the sizes of the associated global data items, and the communication link bandwidths of the machines. The scenarios were generated for different numbers of subtasks and different numbers of machines, as specified below. The estimated expected execution time for each subtask on each machine, the number of global data items, the size of each global data item, and the bandwidth of each input link of each machine were randomly generated with uniform probability over some predefined ranges. For each machine, the bandwidth of the output link is made equal to that of the input link. The producer and consumers of each global data item were also generated randomly. The scenario generation used a $|G| \times |S|$ dependency matrix to guarantee that the precedence constraints from data dependencies were acyclic. Each row of this matrix specified the data dependencies of the corresponding global data item. In each row, the producer must appear to the left of all of its consumers.

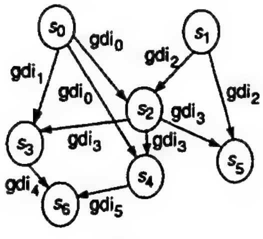
These randomly generated scenarios were used for three reasons: (1) it is desirable to obtain data that demonstrate the effectiveness of the approach over a broad range of conditions, (2) a generally accepted set of HC benchmark tasks does not exist, and (3) it is not clear what characteristics a "typical" HC task would exhibit [WaA96]. Determining a representative set of HC task benchmarks remains a current and unresolved challenge for the scientific community in this research area.

In this research, small-scale and larger scenarios were used to quantify the performance of this GA-based approach. The scenarios were grouped into three categories, namely tasks with light, moderate, and heavy communication loads. A lightly communicating task has its number of global data items in the range of $0 \leq |G| < (1/3)|S|$; a moderately communicating task has its number of global data items in the range of $(1/3)|S| \leq |G| < (2/3)|S|$; and a heavily communicating task has its number of global data items in the range of $(2/3)|S| \leq |G| < |S|$. The ranges of the global data item sizes and the estimated subtask execution times were both from 1 to 1000. For these scenarios, the bandwidths of the input and output links were randomly generated, ranging from 0.5 to 1.5. Hence, the communication times in these scenarios were source and destination machine dependent.

For each scenario, there were many *GA runs*, each of which was a GA search for the best solution to this scenario, starting from a different initial population. The probability of crossover was the same for the matching string and the scheduling string. The probability of mutation was also the same for the matching string and the scheduling string. The stopping criteria were (1) the number of iterations had reached 1000, (2) the population had converged (i.e., all the chromosomes had the same fitness value), or (3) the currently best solution had not improved over the last 150 iterations. All the GA runs discussed in this section had stopped when the best solutions were not improved after 150 iterations.

The GA-based approach was first applied to 20 small-scale scenarios that involved up to ten subtasks, three machines, and seven global data items. The GA runs for small-scale scenarios had the following parameters. The probabilities for scheduling string crossovers, matching string crossovers, scheduling string mutations, and matching string mutations were chosen to be 0.4, 0.4, 0.1, 0.1, respectively. The GA population size, P, for small-scale scenarios was chosen to be 50. For these scenarios, the rank-based roulette wheel selection scheme was used. The angle ratio of the sectors on the roulette wheel for two adjacently ranked chromosomes, R, was chosen to be $1 + 1/P$. By using this simple formula, the angle ratio between the slots of the best and median chromosomes for $P = 50$ (and also for $P = 200$ for larger scenarios discussed later in this section) was very close to the optimal empirical ratio value of 1.5 in [Whi89].

The results from a small-scale scenario were used here to illustrate the search process. This scenario had $|S| = 7$, $|M| = 3$, and $|G| = 6$. The DAG, the estimated execution times, and the transfer times of the global data items are shown in Figs. 12a–12c, respectively. The total numbers of possible different matching strings and different valid scheduling strings (i.e., topological sorts of the DAG) were $3^7 = 2187$ and 16, respectively. Thus, the total search space had 2187 × 16 = 34,992 possible chromosomes.



(a)

	m_0m_1	m_0m_2	m_1m_2
gdi$_0$	489	321	489
gdi$_1$	1244	818	1244
gdi$_2$	62	41	62
gdi$_3$	830	545	830
gdi$_4$	387	255	387
gdi$_5$	999	656	999

(c)

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
m_0	872	251	542	40	742	970	457
m_1	898	624	786	737	247	749	451
m_2	708	778	23	258	535	776	15

(b)

FIG. 12. A small-scale simulation scenario: (a) the DAG, (b) the estimated execution times, and (c) the transfer times of the global data items.

Figure 13 depicts the evolution process of one GA run on this scenario. In each subfigure, the ss axis is the scheduling string axis and the mat axis is the matching string axis. The 16 different scheduling strings on the ss axis are numbered from 1 to 16. The 2187 different matchings on the mat axis are numbered from 1 to 2187. If there is a chromosome at a point (mat, ss), then there is a vertical pole at (mat, ss). The height of a pole represents the quality of the chromosome. The greater the height of the pole, the better a chromosome

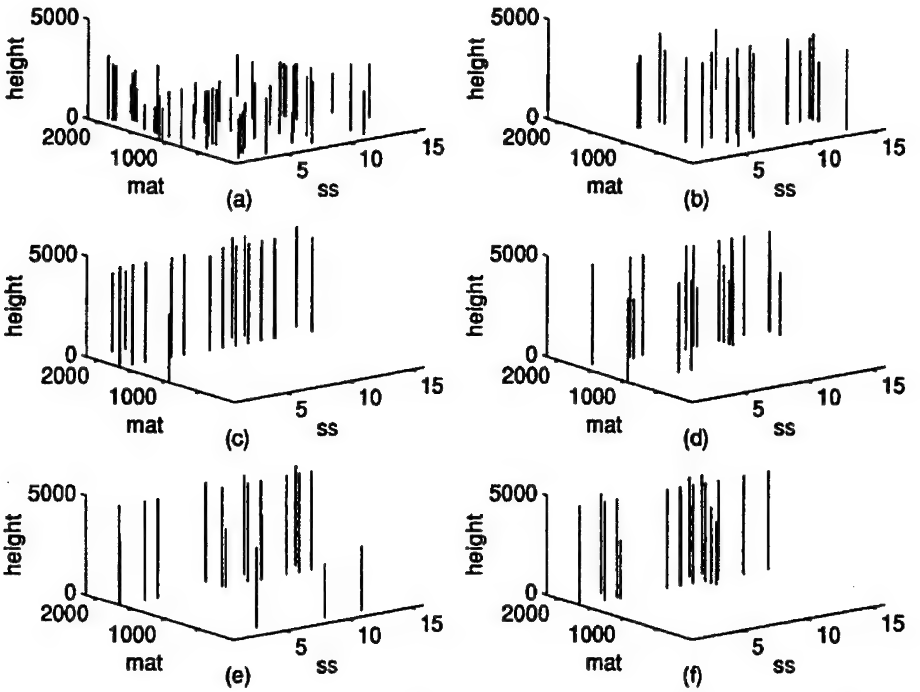


FIG. 13. Evolution of a GA run for the scenario in Fig. 12: (a) at iteration 0, (b) at iteration 40, (c) at iteration 80, (d) at iteration 120, (e) at iteration 160, and (f) at iteration 203 (when the search stopped). Height is a positive constant minus the task execution time associated with *(mat, ss)*.

(solution) is. Multiple identical chromosomes at the same point are not differentiated. Figures 13a–13f show the distributions of the distinct chromosomes at iterations 0, 40, 80, 120, 160, and 203, respectively. This GA run stopped at iteration 203. This GA-based approach found multiple best solutions that have the same completion time, as shown in Fig. 13f.

Exhaustive searches were performed to find the optimal solutions for the small-scale scenarios. For each of the small-scale scenarios that were conducted, the GA-based approach found one or more optimal solutions that had the same completion time, verified by the best solution(s) found by the exhaustive search. The GA search for a small-scale scenario that had ten subtasks, three machines, and seven global data items took about 1 min to find multiple optimal solutions on a Sun Sparc5 workstation while the exhaustive search took about 8 h to find these optimal solutions.

The performance of this GA-based approach was also examined using larger scenarios with up to 100 subtasks and 20 machines. These larger scenarios were generated using the same procedure as for generating the small scenarios. The GA population size for larger scenarios was chosen to be 200.

Larger scenarios are intractable problems. It is currently impractical to directly compare the quality of the solutions found by the GA-based approach for these larger scenarios with those found by exhaustive searches. It is also difficult to compare the performance of different HC task matching and scheduling approaches due to the different HC system models used. Examples of such differences are given in the next section. However, the model used in [IvO95] is similar to the one being used in this research work. Hence, the performance of the GA-based approach on larger scenarios was compared with the nonevolutionary levelized min-time (*LMT*) heuristic proposed in [IvO95].

The LMT heuristic first levelizes the subtasks in the following way. The subtasks that have no input global data items are at the highest level. Each of the remaining subtasks is at one level below the lowest producer of its global data items. The subtasks at the highest level are to be considered first. The LMT heuristic averages the estimated execution times for each subtask across all machines. At each level, a level-average execution time, i.e., the average of the machine-average execution times of all subtasks at this level, is also computed. If there are some levels between a subtask and its closest child subtask, the level-average execution time of each middle level is subtracted from the machine-average execution time of this subtask. The adjusted machine-average execution times of the subtasks are used to determine the priorities of the subtasks within each level; i.e., a subtask with a larger average is to be considered earlier at its level. If the number of subtasks at a level is greater than the number of machines in the HC suite, the subtasks with smaller averages are merged so that as the result, the number of the combined subtasks at each level equals the number of machines available. When a subtask is being considered, it is assigned to the fastest machine available from those machines that have not yet been assigned any subtasks from the same level. Then, it is scheduled using the scheduling principles discussed in Section 9.

Another nonevolutionary heuristic, the *baseline* (*BL*), was developed as part of this GA research and the solution it found was incorporated into the initial population. Similar to the LMT heuristic, the baseline heuristic first levelizes the subtasks based upon their data dependencies. Then all subtasks are ordered such that a subtask at a higher level comes before one at a lower level. The subtasks in the same level are arranged in descending order of their numbers of output global data items (ties are broken arbitrarily). The subtasks are then scheduled in this order. Let the *i*th subtask in this order be σ_i, where $0 \leq i < |S|$. First, subtask σ_0 is assigned to a machine that gives the shortest completion time for σ_0. Then, the heuristic evaluates $|M|$ assignments for σ_1, each time assigning σ_1 to a different machine, with the previously decided machine assignment of σ_0 left unchanged. The subtask σ_1 is finally assigned to a machine that gives the shortest overall completion time for both σ_0 and σ_1. The baseline heuristic continues to evaluate the remaining subtasks in their order to be considered. When scheduling subtask σ_i, $|M|$ possible machine assignments are evaluated, each time with the previously decided machine assignments of subtasks σ_j $(0 \leq j < i)$ left unchanged. Subtask σ_i is finally assigned to a machine that gives the shortest overall completion time of subtasks σ_0 through σ_i. The total number of evaluations is thus $|S| \times |M|$, and only i subtasks (out of $|S|$) are considered when performing evaluations for the $|M|$ machine assignments for subtask σ_i.

Compared with the LMT and baseline nonevolutionary heuristics, the execution time of the GA-based approach was much greater, but it found much better solutions. This is appropriate for off-line matching and scheduling, rather than for real-time use (although in some applications, off-line precomputed GA mapping can be used on-line in real time [BuR97]).

To determine the best GA parameters for solving larger HC matching and scheduling problems, 50 larger scenarios were randomly generated in each communication category. Each of these scenarios contained 50 subtasks and five machines. For each scenario, 400 GA runs were conducted, half of which used the rank-based roulette selection scheme and the other half used the value-based roulette selection scheme. The 200 GA runs using the same selection scheme on each scenario had the following combinations of crossover probability and mutation probability. The crossover probability ranged from 0.1 to 1.0 in steps of 0.1, and the mutation probability ranged from 0.04 to 0.40 in steps of 0.04 and from 0.4 to 1.0 in steps of 0.1. Let the *relative solution quality* be the task completion time of the solution found by the LMT heuristic divided by that found by the approach being investigated. A greater value of the relative solution quality means that the approach being investigated finds a better solution to the HC matching and scheduling problem (i.e., with a shorter overall completion time for the application task represented by the

DAG). With each crossover and mutation probability pair and for each communication load, the average relative solution quality of the 50 GA runs, each on a different scenario, was computed. The following is a brief discussion and comparison of the rank-based and the value-based selection schemes, based on the experimental data obtained. Three-dimensional mesh and two-dimensional contour plots were used to analyze the experimental data. A detailed discussion and comparisons can be found in [Wan97].

Table I lists the best and worst average relative solution quality and the associated probabilities for each communication load with each selection scheme. The data in the table illustrates that the best solution found with the rank-based selection scheme was always better than that found with the value-based selection scheme in each communication load category. An analysis of the GA runs showed that the value-based selection scheme tended to improve the average fitness value of the population faster than the fitness value of the currently best chromosome. This caused the slot angle for the best chromosome in the population to decrease, thus reducing its possibility of selection in the search for better solutions.

For both selection schemes and each communication load category, a *region of good performance* could be identified for a range of crossover and mutation probabilities. The variation in the quality of solutions in each region of good performance was less than 33% of that over the entire range of crossover and mutation probabilities. In every case, this region of good performance also included the best relative solution quality. From Table I, it could be seen that the regions of good performance generally consisted of moderate to high crossover probability and low to moderate mutation probability. The values of the crossover and mutation probabilities in these regions are consistent with the results from the GA literature, which show that crossover is GA's major operator and mutation plays a secondary role in GA searches [Dav91, Gol89, SrP94].

With the rank-based selection scheme the regions of good performance were larger than those with the value-based selection scheme. Hence, the rank-based selection scheme was less sensitive to crossover and mutation probability selections to achieve good performance, whereas with the value-based selection scheme, one had to be careful in choosing crossover and mutation probabilities for the GA to find good solutions to the HC matching and scheduling problem.

Because the rank-based selection found better solutions and it was less sensitive to probability selections for good performance, it was chosen to be used for the larger scenarios. The crossover and mutation probabilities, as listed in Table I, with which the best relative solution quality had been achieved, were used in each corresponding communication load category. When matching and scheduling real tasks, the communication load can be determined by computing the ratio of the number of global data items to the number of subtasks. Once the communication load category is known, a probability pair from the corresponding region of good performance can be used.

TABLE I
Best and Worst Relative Solution Quality Found by the Rank-Based and Value-Based Selection Schemes with Associated Probabilities in Each Communication Load Category

Comm. load	Selection scheme	Best	Worst	Region of good performance
Light	Rank-based	Quality = 2.9138 $P_{\text{xover}} = 0.4$ $P_{\text{mut}} = 0.40$	Quality = 2.4692 $P_{\text{xover}} = 0.5$ $P_{\text{mut}} = 1.00$	Quality = 2.7876 to 2.9138 P_{xover} = 0.4 to 1.0 P_{mut}= 0.20 to 0.40
Light	Value-based	Quality = 2.7328 $P_{\text{xover}} = 0.9$ $P_{\text{mut}} = 0.16$	Quality = 2.2968 $P_{\text{xover}} = 1.0$ $P_{\text{mut}} = 0.90$	Quality = 2.6085 to 2.7328 P_{xover} = 0.6 to 0.9 P_{mut}= 0.12 to 0.24
Moderate	Rank-based	Quality = 2.7451 $P_{\text{xover}} = 0.5$ $P_{\text{mut}} = 0.36$	Quality = 2.1520 $P_{\text{xover}} = 0.7$ $P_{\text{mut}} = 1.00$	Quality = 2.5501 to 2.7451 P_{xover} = 0.3 to 1.0 P_{mut} = 0.20 to 0.50
Moderate	Value-based	Quality = 2.4424 $P_{\text{xover}} = 0.9$ $P_{\text{mut}} = 0.12$	Quality = 1.9615 $P_{\text{xover}} = 1.0$ $P_{\text{mut}} = 1.00$	Quality = 2.2958 to 2.4424 P_{xover} = 0.5 to 1.0 P_{mut} = 0.04 to 0.24
Heavy	Rank-based	Quality = 2.3245 $P_{\text{xover}} = 1.0$ $P_{\text{mut}} = 0.20$	Quality = 1.7644 $P_{\text{xover}} = 0.1$ $P_{\text{mut}} = 1.00$	Quality = 2.1568 to 2.3245 P_{xover} = 0.6 to 1.0 P_{mut}= 0.16 to 0.40
Heavy	Value-based	Quality = 2.0883 $P_{\text{xover}} = 0.6$ $P_{\text{mut}} = 0.20$	Quality = 1.6598 $P_{\text{xover}} = 1.0$ $P_{\text{mut}} = 1.00$	Quality = 1.9582 to 2.0883 P_{xover} = 0.5 to 1.0 P_{mut} = 0.16 to 0.24

Note. For each communication load category with each selection scheme, the rectangular region of good performance with the boundary crossover and mutation probabilities are listed. The best and worst relative solution quality within each region are also shown. In the table, P_{xover} is the crossover probability and P_{mut} is the mutation probability.

On Sun Sparc5 workstations, for these larger scenarios, both the LMT heuristic and the baseline heuristic took no more than 1 min of CPU time to execute. The average CPU execution time of the GA-based approach on these scenarios ranged from less than 1 min for the smallest scenarios (i.e., five subtasks, two machines, and light communication load) to about $3\frac{1}{2}$ h for the largest scenarios (i.e., 100 subtasks, 20 machines, and heavy communication load). Recall that it is assumed that this GA-based approach will be used for application tasks that are large production jobs such that the one time investment of this high execution time is justified.

The performance of the GA-based approach was also compared with that of a random search. For each iteration of the random search, a chromosome was randomly generated. This chromosome was evaluated and the fitness value was compared with the saved best fitness value. If the fitness value of the current chromosome was better than the saved best value, it became the saved best fitness value. For each scenario, the random search iterated for the same length of time as that taken by the GA-based approach on the same scenario.

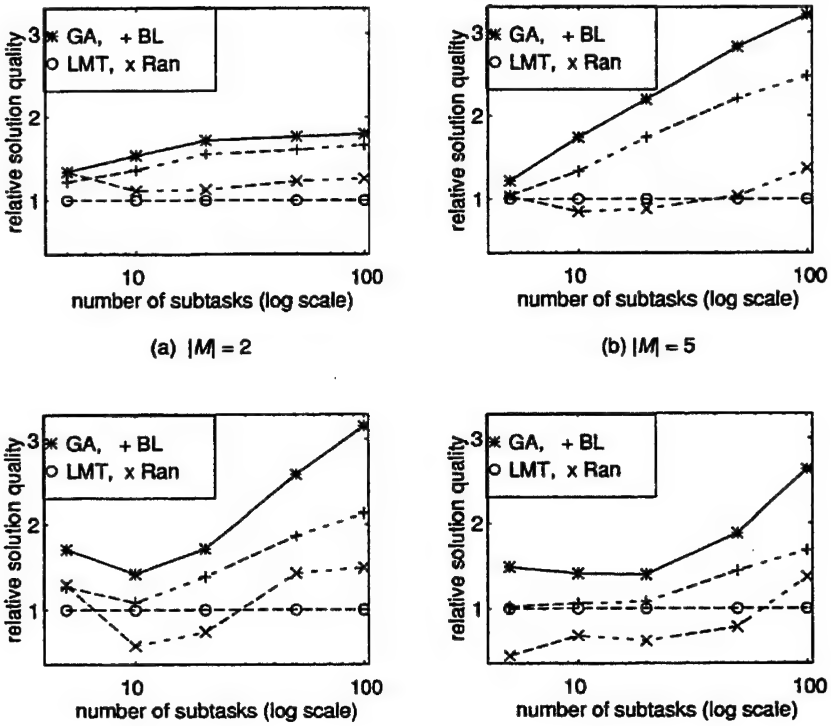
Figure 14 shows the performance comparisons between the LMT heuristic and the GA-based approach for lightly communicating larger scenarios. In the figure, the horizontal axes are the number of subtasks in log scale. The vertical axes are the relative solution quality of the GA-based approach. The relative solution quality of the baseline (BL) heuristic and the random search is also shown in this figure. Each point in the figure is the average of 50 independent scenarios. The performance comparisons among the GA-based approach, the LMT heuristic, the baseline heuristic, and the random search for moderately communicating and heavily communicating larger scenarios are shown in Figs. 15 and 16, respectively.

In all cases, the GA-based approach presented here outperformed these other two heuristics and the random search. The improvement of the GA-based approach over the others showed an overall trend to increase as the number of subtasks increased. The exact shape of the GA-based-approach performance curves is not as significant as the overall trends because the curves are for a heuristic operating on randomly generated data, resulting in some varied performance even when averaged over 50 scenarios for each data point.

11. RELATED WORK

Different approaches to the HC matching and scheduling problem are difficult to compare. One of the reasons is that the HC models used vary from one approach to another. Furthermore, as discussed in Section 10, established test benchmarks do not exist at this time.

The most related research using GAs for HC includes [ShW96, SiY96, TiP96]. Our research significantly differs from the above approaches in terms of the HC models assumed. The following is a brief discussion of the related research work.



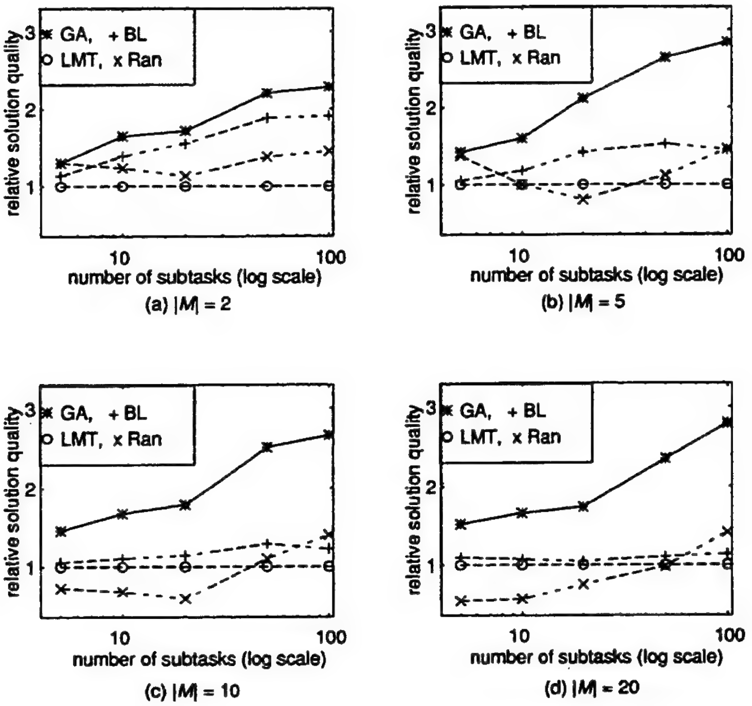
(a) |*M*| = 2

(b) |*M*| = 5

(c) |*M*| = 10

(d) |*M*| = 20

FIG. 14. Performance comparisons of the GA-based approach relative to the LMT heuristic for lightly communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic and the random search are also shown.



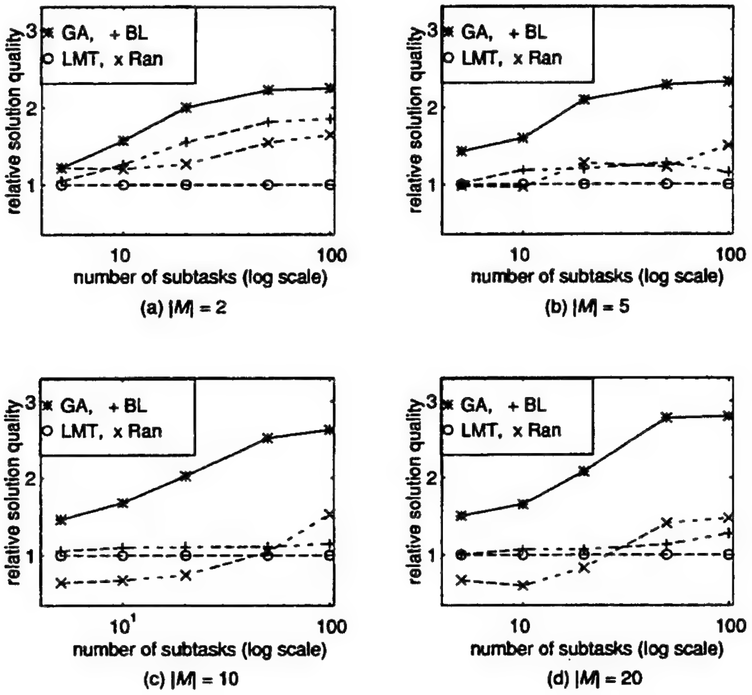
(a) |*M*| = 2

(b) |*M*| = 5

(c) |*M*| = 10

(d) |*M*| = 20

FIG. 15. Performance comparisons of the GA-based approach relative to the LMT heuristic for moderately communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite.The relative performance of the baseline heuristic and the random search are also shown.



(a) |*M*| = 2

(b) |*M*| = 5

(c) |*M*| = 10

(d) |*M*| = 20

FIG. 16. Performance comparisons of the GA-based approach relative to the LMT heuristic for heavily communicating larger scenarios in (a) a two-machine suite, (b) a five-machine suite, (c) a ten-machine suite, and (d) a 20-machine suite. The relative performance of the baseline heuristic and the random search are also shown.

In [SiY96], a GA-based approach was proposed, in which the matcher/scheduler can utilize an unlimited number of machines as needed. In our proposed approach, however, an HC suite of a fixed number of machines is assumed. Another difference between these two approaches is that in [SiY96] a machine can send and receive data to and from an unlimited number of different machines concurrently. In our proposed approach, it is assumed that each machine has a single input link and a single output link such that all the input communications to one machine have to be serialized and all the output communications from one machine have to be serialized. A third difference between these two approaches is that in [SiY96] data can only be obtained from the original producer. In our proposed approach, however, data can be obtained either from the producer or from another subtask that has received the data. This is the data forwarding situation that was discussed in more detail in Section 9. Unlike the chromosome structure used in our proposed approach, which represents both matching and scheduling decisions, in [SiY96], a chromosome structure that only has the matching decision was used. Because of the assumptions made in [SiY96], for each matching decision an optimal scheduling can be computed.

Although a fully connected interconnection network is assumed in both [ShW96] and our proposed approach, in [ShW96] each machine can send to and receive from an unlimited number of different machines concurrently. Data forwarding is not utilized in [ShW96]. A simulated annealing technique was used in [ShW96] to do the chromosome selection. Similar to [SiY96], a chromosome structure that only has the matching decision was also used in [ShW96]. A nonrecursive algorithm was used in [ShW96] to determine a scheduling for each matching decision.

A GA-based approach in [TiP96] was used to design application-specific multiprocessor systems. Different from the goal set for this research, which is to minimize the total execution time, [TiP96] considered both the execution time and the system cost for a given application. In our approach, however, it is assumed that a machine suite is given, and the only goal is to minimize the completion time of the application.

12. CONCLUSION

A novel genetic-algorithm-based approach for task matching and scheduling in HC environments was presented. This GA-based approach can be used in a variety of HC environments because it does not rely on any specific communication subsystem models. It is applicable to the static scheduling of production jobs and can be readily used for scheduling multiple independent tasks (and their subtasks) collectively. For small-scale scenarios, the proposed approach found optimal solutions. For larger scenarios, it outperformed two nonevolutionary heuristics and a random search.

There are a number of ways this GA-based approach for HC task matching and scheduling may be built upon for future research. These include extending this approach to allow multiple producers for each of the global data items, parallelizing the GA-based approach, developing evaluation procedures for other communication subsystems, and considering loop and data-conditional constructs that involve multiple subtasks.

In summary, a novel GA design was developed for use in HC. This GA design has been shown to be a viable approach to the important problems of matching and scheduling in an HC environment.

ACKNOWLEDGMENTS

The authors thank M. Maheswaran, J. M. Siegel, M. D. Theys, and S. Wang for their valuable comments. A preliminary version of portions of this work was presented at the 5th Heterogeneous Computing Workshop (HCW'96).

REFERENCES

[AhD96] Ahmad, I., and Dhodhi, M. K. Multiprocessor scheduling in a genetic paradigm. *Parallel Comput.* **22,** 3 (Mar. 1996), 395–406.

[BeS94] Benten, M. S. T., and Sait, S. M. Genetic scheduling of task graphs. *Internat. J. Electron.* **77,** 4 (Apr. 1994), 401–405.

[BuR97] Budenske, J. R., Ramanujan, R. S., and Siegel, H. J. On-line use of off-line derived mappings for iterative automatic target recognition tasks and a particular class of hardware platforms. *Proc. 1997 Heterogeneous Computing Workshop (HCW'97).* IEEE Computer Society, Geneva, Switzerland, 1997, pp. 96–110.

[ChL88] Chen, C. L., Lee, C. S. G., and Hou, E. S. H. Efficient scheduling algorithms for robot inverse dynamic computation on a multiprocessor system. *IEEE Trans. Systems Man Cybernet.* **18,** 5 (Sept.–Oct. 1988), 729–743.

[CoL92] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1992.

[Dav91] Davis, L. (Ed.). *Handbook of Genetic Algorithms.* Van Nostrand–Reinhold, New York, 1991.

[EsS94] Eshaghian, M. M., and Shaaban, M. E. Cluster-M programming paradigm. *Internat. J. High Speed Comput.* **6,** 2 (Feb. 1994), 287–309.

[Fer89] Fernandez-Baca, D. Allocating modules to processors in a distributed system. *IEEE Trans. Softare Engrg.* **SE-15,** 11 (Nov. 1989), 1427–1436.

[Fre89] Freund, R. F. Optimal selection theory for superconcurrency. *Proc. Supercomputing '89.* IEEE Computer Society, Reno, NV, 1989, pp. 699–703.

[Fre94] Freund, R. F. The challenges of heterogeneous computing. *Parallel Systems Fair at the 8th International Parallel Processing Symp.* IEEE Computer Society, Cancun, Mexico, 1994, pp. 84–91.

[FrS93] Freund, R. F., and Siegel, H. J. Heterogeneous processing. *IEEE Comput.* **26,** 6 (June 1993), 13–17.

[GhY93] Ghafoor, A., and Yang, J. Distributed heterogeneous supercomputing management system. *IEEE Comput.* **26,** 6 (June 1993), 78–86.

[Gol89] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley, Reading, MA, 1989.

[HoA94] Hou, E. S. H., Ansari, N., and Ren, H. Genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Systems* **5,** 2(Feb. 1994), 113–120.

[Hol75] Holland, J. H. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, 1975.

[HoT89] Hoebelheinrich, R., and Thomsen, R. Multiple crossbar network integrated supercomputing framework. *Proc. Supercomputing '89.* IEEE Computer Society and ACM SIGARCH, Reno, NV, 1989, pp. 713–720.

[IvO95] Iverson, M. A., Ozguner, F., and Follen, G. J. Parallelizing existing applications in a distributed heterogeneous environment.

Proc. 1995 Heterogeneous Computing Workshop (HCW'95). IEEE Computer Society, Santa Barbara, CA, 1995, pp. 93–100.

[KhP93] Khokhar, A., Prasanna, V. K., Shaaban, M., and Wang, C. L. Heterogeneous computing: Challenges and opportunities. *IEEE Comput.* **26,** 6 (June 1993), 18–27.

[MuC69] Muntz, R. R., and Coffman, E. G. Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comput.* **C-18,** 11 (Nov. 1969), 1014–1020.

[NaY94] Narahari, B., Youssef, A., and Choi, H. A. Matching and scheduling in a generalized optimal selection theory. *Proc. 1994 Heterogeneous Computing Workshop (HCW'94).* IEEE Computer Society, Cancun, Mexico, 1994, pp. 3–8.

[RiT94] Ribeiro Filho, J. L., and Treleaven, P. C. Genetic-algorithm programming environments. *IEEE Comput.* **27,** 6 (June 1994), 28–43.

[Rud94] Rudolph, G. Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Networks* **5,** 1 (Jan. 1994), 96–101.

[ShW96] Shroff, P., Watson, D. W., Flann, N. S., and Freund, R. F. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 98–104.

[SiA96] Siegel, H. J., Antonio, J. K., Metzger, R. C., Tan, M., and Li, Y. A. Heterogeneous computing. In Zomaya, A. Y. (Ed.). *Parallel and Distributed Computing Handbook.* McGraw-Hill, New York, 1996, pp. 725–761.

[SiY96] Singh, H., and Youssef, A. Mapping and scheduling heterogeneous task graphs using genetic algorithms. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 86–97.

[SrP94] Srinivas, M., and Patnaik, L. M. Genetic algorithms: A survey. *IEEE Comput.* **27,** 6 (June 1994), 17–26.

[Sun92] Sunderam, V. S. Design issues in heterogeneous network computing. *Proc. Workshop on Heterogeneous Processing.* IEEE Computer Society, Beverly Hills, CA, March 1992, pp. 101–112.

[TaA95] Tan, M., Antonio, J. K., Siegel, H. J., and Li, Y. A. Scheduling and data relocation for sequentially executed subtasks in a heterogeneous computing system. *Proc. 1995 Heterogeneous Computing Workshop (HCW'95).* IEEE Computer Society, Santa Barbara, CA, April 1995, pp. 109–120.

[TiP96] Tirat-Gefen, Y. G., and Parker, A. C. MEGA: An approach to system-level design of application-specific heterogeneous multiprocessors. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 105–117.

[ToR93] Tolmie, D., and Renwick, J. HiPPI: Simplicity yields success. *IEEE Network* **7,** 1 (Jan. 1993), 28–32.

[WaA94] Watson, D. W., Antonio, J. K., Siegel, H. J., and Atallah, M. J. Static program decomposition among machines in an SIMD/SPMD heterogeneous environment with nonconstant mode switching costs. *Proc. 1994 Heterogeneous Computing Workshop (HCW'94).* IEEE Computer Society, Cancun, Mexico, April 1994, pp. 58–65.

[WaA96] Watson, D. W., Antonio, J. K., Siegel, H. J., Gupta, R., and Atallah, M. J. Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment. *Proc. 1996 Heterogeneous Computing Workshop (HCW'96).* IEEE Computer Society, Honolulu, HI, April 1996, pp. 24–37.

[Wan97] Wang, L. A genetic-algorithm-based approach for subtask matching and scheduling in heterogeneous computing environments and a comparative study on parallel genetic algorithms. Ph.D. thesis, School of Electrical and Computer Engineering, Purdue University, 1997.

[WeW94] Weems, C. C., Weaver, G. E., and Dropsho, S. G. Linguistic support for heterogeneous parallel processing: A survey and an approach. *Proc. 1994 Heterogeneous Computing Workshop (HCW'94).* IEEE Computer Society, Cancun, Mexico, April 1994, pp. 81–88.

[Whi89] Whitley, D. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. *Proc. 1989 International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA, June 1989, pp. 116–121.

Received February 1, 1997; received July 15, 1997; revised August 25, 1997

LEE WANG received a B.S.E.E. degree from Tsinghua University, Beijing, China; an M.S. degree in physics from Bowling Green State University, Bowling Green, Ohio, USA; and a Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, Indiana, USA, in 1990, 1992, and 1997, respectively. He worked as a research engineer for Architecture Technology Corporation during the summer of 1996. In June 1997, Dr. Wang joined Microsoft Corporation in Redmond, Washington, USA. His research interests include task matching and scheduling in heterogeneous computing environments, parallel languages and compilers, reconfigurable parallel computing systems, data parallel algorithms, distributed operating systems, and multimedia technology development. Dr. Wang has authored or coauthored one journal paper, six conference papers, two book chapters, and one language user's manual.

HOWARD JAY SIEGEL received B.S.E.E. and B.S. Management degrees from MIT (1972), and the M.A. (1974), M.S.E. (1974), and Ph.D. (1977) degrees in computer science from Princeton University. He is a Professor in the School of Electrical and Computer Engineering at Purdue University. His research interests include heterogeneous computing, parallel processing, interconnection networks, and the design and use of the PASM reconfigurable parallel machine. He is an IEEE Fellow (1990) and an ACM Fellow (1998), has coauthored more than 240 technical papers, was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*, and is a member of the Steering Committee of the annual Heterogeneous Computing Workshop sponsored by the IEEE Computer Society.

VWANI P. ROYCHOWDHURY received the B.Tech. degree from the Indian Institute of Technology, Kanpur, India, and the Ph.D. degree from Stanford University, Stanford, CA, in 1982 and 1989, respectively, all in Electrical Engineering. He is currently a professor in the Department of Electrical Engineering at the University of California, Los Angeles. From August 1991 to June 1996, he was a faculty member at the School of Electrical and Computer Engineering at Purdue University. His research interests include parallel algorithms and architectures, design and analysis of neural networks, application of computational principles to nanoelectronics, special purpose computing arrays, VLSI design, and fault-tolerant computation. He has coauthored several books including "Discrete Neural Computation: A Theoretical Foundation" (Prentice-Hall, Englewood Cliffs, NJ, 1995) and "Theoretical Advances in Neural Computation and Learning" (Kluwer Academic, Dordrecht, 1994).

ANTHONY A. MACIEJEWSKI received the B.S.E.E., M.S., and Ph.D. degrees in electrical engineering from The Ohio State University, Columbus, OH, in 1982, 1984, and 1987, respectively. In 1985–1986 he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan. Since 1988 he has been with the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana, where he is currently an Associate Professor. His primary research interests center on the simulation and control of kinematically redundant robotic systems.

Proceedings of *Fourteenth Computer Security Applications Conference*, Phoenix, AZ. pp. 167--77, December 1998.

Security Architecture for a Virtual Heterogeneous Machine

Roger Wright
Computer Science Dept.
Naval Postgraduate School
Monterey, CA 93943
rewright@nps.navy.mil

David J. Shifflett
Rolands and Associates Corp.
500 Sloat Avenue
Monterey, CA 93940
dshifflett@gwdi.com

Cynthia E. Irvine
Computer Science Dept.
Naval Postgraduate School
Monterey, CA 93943
irvine@cs.nps.navy.mil

Abstract

We describe security for a virtual heterogeneous machine (VHM). Our security architecture is based upon separation of services into four distinct domains. It is designed to take advantage of operating system support for domains, where available. We have chosen to use emerging public key technology as an interim solution to provide domain separation. A prototype demonstration of our architecture has been developed.

1. Introduction

Science, engineering and defense are witnessing the emergence of a new approach to computation. It depends not on single isolated centers of excellence for high performance computing, but upon a network of computers bound together by an overlying framework that presents to users a powerful virtual heterogeneous machine (VHM) intended to support high throughput computing [19]. Applications may consist of a single thread of execution, a set of sequential jobs, or may be parallelized across several components of the underlying system. Some applications may require a variety of computational and storage resources ranging from desktop PCs for the submission of jobs to supercomputers for computationally intensive tasks.

Efforts are currently underway to define the VHMs which will manage user tasks. A major research area for these systems is the formulation of adaptive scheduling techniques based on task and network properties. Ultimately, the technologies emerging from the creation of an advanced computational infrastructure will be integrated into systems supporting finance, industry, and individual users. In all cases, security is a concern for these systems.

The diversity of the user population employing a VHM implies that one security solution may not be appropriate for all individuals or even for all tasks of one particular individual. We are collaborating in a research project to investigate the properties and services of a VHM and are examining security concerns and technologies in that context. This paper is a report on the progress of that effort.

We will describe how notions of privilege and system integrity can be incorporated into an architecture intended to execute in a heterogeneous computing environment. The remainder of the paper is organized as follows. Section 2. outlines the objectives and high level architecture of the VHM to which our protection architecture is being applied. Section 3. describes the security requirements for the system. Section 4. describes the security architecture we have developed for the VHM. Our prototype implementation and its use of the Intel Common Data Security Architecture, is discussed in section 5. Section 6. will contrast our work with that of several other VHM security efforts and will describe our future plans. Section 7. provides a summary.

2. Management System for Heterogeneous Networks

The intent of the Management System for Heterogeneous Networks (MSHN, pronounced "mission"') is to construct a virtual heterogeneous machine designed to run as an application on a variety of operating systems and hardware platforms [11]. It will provide end-to-end support for applications in distributed and heterogeneous, shared environments. In addition to supporting compute-intensive jobs, MSHN is intended to provide a responsive and flexible execution environment for real-time, interactive, and I/O-intensive tasks. When heterogeneous resources are

This work was supported in part by the DARPA/ITO Quorum Program.

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

simultaneously shared by several applications, each with its own unique quality of service (QoS) requirement, MSHN will be able to efficiently assign resources to the applications so that they will attain their requested QoS.

Each *job* is characterized by code and a dataset. An instance of a job will be the code and a particular dataset and is associated with a particular user. For each job, MSHN will maintain historical data describing the job's *compute characteristics*. When a new request to execute the job with a particular dataset is issued, the scheduler will use the job's compute characteristics as well as dynamic information about network and resource loading. The scheduler will make decisions regarding resource allocation to maximize the QoS for a particular job. For example, suppose job A has historically run in 10 minutes on resource X and in 20 minutes on resource Y. When a request to execute the job is issued the scheduler may decide to wait five minutes so that the job can be run on system X rather than schedule it immediately on system Y. In an open environment, even when using the same resource, consistent QoS for jobs may not be guaranteed. Network congestion and load at the compute resource can impact job performance.

Use of MSHN is optional. In theory, users could attempt to decide where a job might best be executed, however, several projects [14][5] have demonstrated that a scheduling and job management service can provide significant benefits, just as traditional operating systems support applications through local resource management services.

These early resource management system demonstrations lacked the ability to address quality of service requirements in a highly distributed and dynamic heterogeneous environment. MSHN will address these issues. It will monitor the load on a large pool of resources and will advise jobs as to the best strategy for achieving QoS objectives for a particular run. It is anticipated that, despite the extra processing required for MSHN to provide monitoring and advisory services, the performance advantages will outweigh the slight computational tax imposed by the VHM. User involvement in the scheduling and execution of jobs might be streamlined through the use of a special MSHN shell that would hide MSHN processing.

2.1 MSHN VHM

The proposed MSHN VHM [15] consists of five major components, as depicted in Figure 1: a user client, three MSHN core services, and a client on each computational server. (The Audit Server will be discussed in a subsequent section.) An assumption of the MSHN architecture is that the physical machines running MSHN components are not necessarily dedicated to MSHN processing. Thus, jobs on a computational server may be divided into two categories: those taking advantage of the services provided by MSHN and those that do not.

Client libraries will "wrap" user jobs at the application origination points and at the compute resources. These libraries are intended to provide the services of the MSHN VHM transparently: legacy application code will require no modification to be executed in the MSHN environment. Adaptive applications will benefit from environmental information that may be signalled to them through the MSHN services.

The Resource Requirements Database is used to store initial estimates of per job requirements and to maintain a record of the resources that have historically been consumed by a particular job. This may include the level of service requested by the users as well as priorities for particular classes of applications. As experiential data are accumulated, the resource requirements for the execution of future instances of the job are adjusted.

The Resource Status Server is the most dynamic of those maintained by MSHN. Here information regarding the resource consumption by all currently executing jobs on the VHM is maintained. In addition, client library monitors associated with the network and computational resources provide up-to-date information regarding the availability of resources on the VHM.

The Scheduling Server maintains a set of algorithms used to produce advisory schedules for jobs. If the Resource Status Server flags a job as being bogged down, a request may be issued to the scheduler to seek a better scheduling strategy for the task. That strategy may include task checkpointing and relocation [16].

A MSHN project goal is to provide support not only for traditional static applications, but to enhance the performance characteristics of dynamic and adaptive applications. This is particularly the case in environments where the load on resources is not predictable, but perhaps best characterized probabilistically. Consider, for example, an adaptive application in which a user asks for time-critical processing that will return high precision and high fidelity results. As the application is executed, MSHN will use the client libraries associated with all jobs on the VHM to monitor the resources of the VHM, quantifying and accounting for uncertainty throughout the distributed processing mechanism. Intermediate reports on job status and resource usage are transmitted

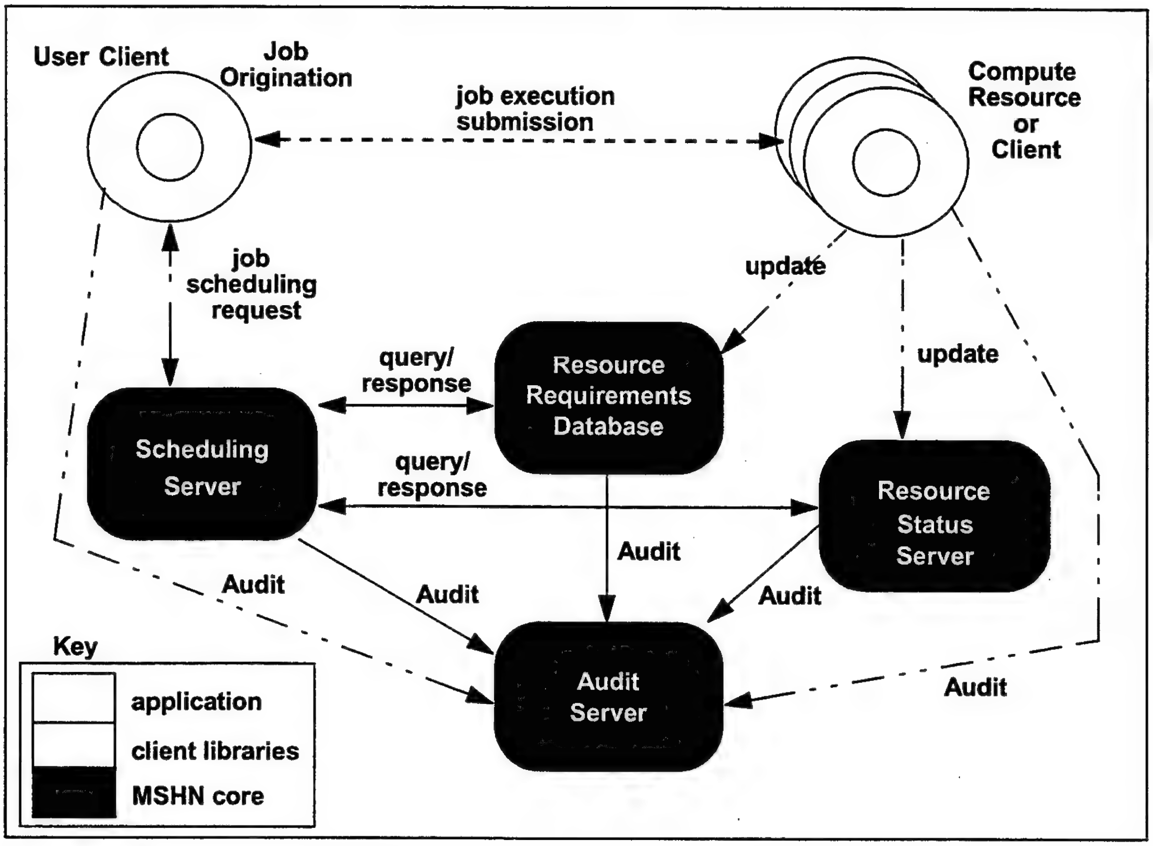


Figure 1. MSHN Architecture

by the client libraries to the Resource Status Server. Should available resources be insufficient to achieve the original time, precision and fidelity objectives, the MSHN Scheduler will trigger a notification to the client library of the affected application. In the case of an adaptive application, the application can adapt to the changed environment, thus satisfying modified, and realistically achievable QoS objectives. Special client libraries could provide adaptivity services to legacy applications. Upon completion of the task, results are returned to the user, while a summary of resource usage by the job is recorded in the Resource Requirements Database.

MSHN jobs consist of both code and the data set to be processed by that code. The need to define the job both in terms of its code and data set, or at least a characterization of the data set, arises from quality of service objectives for each job. Consider for example, code that will compute Fast Fourier Transforms (FFTs) on seismic data. Clearly the granularity and duration of the data will affect the amount of processing required. Hence, despite the fact that the code applied to both a short event and a long one is the same, the jobs differ in terms of the computational resources they will consume. As noted above, a variety of QoS requirements may also be associated with a job. Of course, a group of security requirements may also be bound to an instance of a job as a result of the particular code or data involved. A detailed discussion of application security requirements will be given in the next section.

3. Security Requirements

The MSHN security mechanisms are intended for an environment in which users elect to use MSHN services to enhance the quality of service they receive on their jobs. Users request advisory schedules from MSHN core services. When the schedule is returned, jobs are submitted to compute resources under user control and executed in the context of user accounts. Dynamic and summary job status information is reported to MSHN core services via client libraries which "wrap" the user's unaltered job. During job execution, the MSHN scheduler may send revised scheduling information to the client libraries. Under certain conditions, a job may be moved to a different compute resource. If the job is adaptive, information relayed by the MSHN scheduler permits the job to modify its runtime characteristics in order to achieve desired quality of service goals.

This sketch of MSHN activity shows that the VHM depends upon communication between distributed hosts: those providing MSHN core services and those executing client libraries on behalf of users. In the remainder of this section we will present our approach to providing for the integrity of MSHN core components and client services within the networked environment. In addition, we will describe how the MSHN core supports security requirements such as integrity and confidentiality for the jobs it manages.

3.1 General Constraints and Assumptions

Certain MSHN design goals affect several choices regarding the security architecture. These are:

- Use of MSHN is optional. An advisory schedule obtained from the MSHN Scheduler is optional. A user may still choose to execute a job using MSHN clients at compute resources, but the compute resources selected by the user may not correspond to those suggested by the MSHN Scheduler.
- MSHN will not require modification of existing applications.
- MSHN will not require modification to the underlying operating system of any system it uses. This includes MSHN user clients, servers hosting MSHN core components, and MSHN services on compute resources.
- No MSHN component will have arbitrary control over any host. By this we mean that execution of MSHN will not require supervisory or, in Unix terminology, "root" privilege.
- It is assumed that users do not wish to subvert their own jobs with bogus run-time parameters, which would also result in corruption of the Resource Requirements Database. (Users can, of course, always elect to execute their jobs without the benefit of MSHN services.)
- User jobs are not required to reside permanently at compute resources. Code and data can either be bundled and sent to compute resources with the request for execution. Eventually, the MSHN client libraries are envisioned to be capabile of fetching code and data for a job from appropriate repositories.

Assumptions applicable to the security architecture are:

- A user will not attempt to corrupt his own jobs. If a user is concerned that his jobs will be corrupted by other jobs he may be executing on a compute resource, then the best remedy is to establish a separate principle for the execution of jobs managed by MSHN. The underlying operating system, through its mechanisms for the separation of user processes is expected to support this objective.
- Users will be required to log in when accessing compute resources, thus execution on resources will be by a principle representing the user rather than the MSHN core.
- For each user, MSHN will maintain a list of compute resources for which the user is authorized.
- MSHN will rely upon the existence of a public key infrastructure (PKI). The security architecture depends upon the existence and integrity of standard directory services such as X.500 or LDAP and the provision of standardized certificates such as those defined by X.509 v3. All elements of the MSHN architecture must have facilities to support certificate-based public key technology. Support for symmetric key cryptography is also required in most applications of public key cryptography.
- It is assumed that the MSHN client libraries do not contain malicious code.
- Compute resources will be known to the MSHN core. The list of resources will be administratively updated.

3.2 Policy

The security policy we wish to enforce for the MSHN components is as follows.

- Only authorized elements of the MSHN core services, such as the MSHN Scheduler, can query the Resource Status Server and the Resource Requirements Database for job status information. (Authentication and Access Control)
- Communications between the elements of the MSHN core services should be protected from unauthorized modification and disclosure. (Communications Integrity and Confidentiality)
- Status and summary information provided by client libraries to the Resource Status Server and Resource Requirements Database should be identifiable to the granularity of a particular run of a user job. This will prevent bogus updates to MSHN databases by malicious jobs outside of MSHN's purview. It will also permit the identification of rogue jobs which may have been corrupted and are misbehaving. (Authentication, Access Control and Denial of Service)
- Status and summary information provided by client

libraries to the MSHN core databases should be protected from unauthorized modification and disclosure while in transit. (Communications Integrity and Confidentiality)

- Dynamic scheduling advice provided by the MSHN Scheduler to client libraries supporting jobs at compute resources should be identifiable on a per job basis. It is not sufficient for the client library to know that scheduling information has, in fact, come from the MSHN Scheduler. In addition, the client library must be able to ascertain that the scheduling information is intended for the particular job being supported by the client library. (Authentication)
- Dynamic scheduling advice provided by the MSHN Scheduler to client libraries supporting jobs at compute resources should be protected from unauthorized modification and disclosure while in transit. (Communications Integrity and Confidentiality)
- Information maintained in MSHN core databases should be available only to authorized users. Some earlier resource management systems [14] had no notion of access control. The emphasis in these efforts was on scheduling and performance, not security, so everyone was privileged and had access to all job status information.

  A mechanism must be implemented to limit access to job status information. It is expected that there may be tension between limiting access to this information and providing statistically significant accumulations of runs to improve scheduling decisions.
- For each job, historical data on that job can be restricted to a set of authorized users. At first glance, this does not appear to be very hospitable. Why should historical information be restricted when it will help everyone run the job better? Suppose there is a job used for battle planning and that its execution statistics are stored in the Resource Requirements Database. Also suppose that an enemy wants to find out about this job. He could pretend to be submitting a request to run the battle planning job and would obtain from the scheduler advice on where to position his run. Even though he might not have a copy of the code and data, the historical data could yield information regarding the best systems to use for executing the job, duration of the run, and other exploitable information.

  Thus, in the interest of encouraging use of MSHN, a user can always request that MSHN provide a schedule for his instance of the job. Whether that schedule reflects historical information would depend upon the user's access to the historical data associated with other users' runs.

3.3 Accountability

Individual accountability is essential in any system intended to securely process information on behalf of users. Accountability permits liability to be established in the case of violations of policy. If the objective is to establish liability such that the perpetrator of an illicit act can be punished, then accountability must be sufficient to provide proof that the accused perpetrator is at fault.

Since MSHN is running as an application on operating systems that may or may not provide an adequate foundation for our security objectives, we must temper our accountability requirements. Corruption of an underlying operating system lacking penetration resistance could result in corruption of MSHN accountability mechanisms. It would be difficult to send someone to prison for violating MSHN security policy and it would be difficult to successfully litigate contractual agreements based upon the chain of assurance and accountability evidence provided by a VHM dependent on intrinsically weak platforms. So accountability in MSHN can act as a deterrent to abuse by relatively law abiding users, but it will not stop the determined opponent.

Accountability in MSHN will depend upon the identification and authentication of users, first at their local clients and subsequently to the MSHN core service and compute servers. Authentication and audit will provide the mechanisms to tie the user's identification to the system activities he has invoked and create a record of those activities considered security relevant.

Non-repudiation is described as a mechanism such that the sender of a communication is unable to deny that the message was sent and the receiver is unable to deny its receipt. The audit mechanism can serve as a trusted third party providing non-repudiation services for MSHN users. Clearly, a third party beyond MSHN is needed if non-repudiation services are required for transactions between MSHN and its users.

3.4 Assurance

Security enforced for MSHN components does not depend upon enforcement mechanisms built into MSHN itself, but instead upon underlying policy enforcement mechanisms that may be provided by operating systems and perhaps high assurance platforms. This is in concert with the philosophy [12] that more privileged mechanisms are

- more likely to embody the fundamental notions of the reference monitor concept [4] and

- more likely to have undergone scrutiny by qualified and unbiased personnel able to assess the assurance claims made about the mechanism.

The security architecture for the VHM will be designed to take advantage of mechanisms provided by more privileged underlying components. Our objective is to provide the intermediate interfaces necessary so that underlying medium to high assurance mechanisms can be used to protect communications associated with the VHM and user jobs as information is moved across unprotected networks.

We wish to prevent unauthorized updates to critical MSHN databases. Suppose that MSHN core components execute on dedicated servers. Then the likelihood of corruption of MSHN core databases by malicious applications is substantially reduced as is the possibility of exfiltraton of information by malicious software. In such an environment, the use of session keys provides a reliable mechanism to bind the components into a virtual domain.

How does the notion of privilege work in MSHN? Ideally, we would like to have each platform in the VHM provide a minimum of four protection domains: one for the user's application, one for MSHN client resource libraries and services, and one for MSHN protection services. Layered beneath these three domains would be more privileged domains associated with the underlying OS and its libraries. We envision certain essential public key technologies located in highly privileged domains. These include the key distribution centers and the certificate authorities.

4. MSHN Security Architecture

MSHN security is based upon the establishment of the following domains:

1. MSHN Core Domain
2. Client Library Domain
3. Application Domain

Each domain will authenticate itself to other domains using certificate-based technology. The basis for believing the certificates is the integrity provided at

Keys	Description	Key Type	Creator	User	Key Life	Purpose
K_m	MSHN Core Session Key	S	MSHN Core Master	MSHN Core Servers	per core instance	• high speed transmission of message traffic between core components
K_j	Job Session Key	S	MSHN Core	Client Library of job	per task	• identifies MSHN-relevant communications to MSHN Core • transmit audit records • authentication of per-job status information to MSHN Core
K_c	Client Library Session Key	S	MSHN Core	Client Library of Job	per task	• per-task intra-client domain management functions
K_a	Application Session Key	S	MSHN Core	Task Clients on behalf of application	per task	• application-level task communications • receive job execution results
$C_{pub/pri}$	MSHN Core Server Key	P	KDC or user	MSHN servers; one per server	KDC defined	• permits clients to authenticate communications to MSHN servers • core session key distribution • digital signatures
$R_{pub/pri}$	Computer Resource Client	P	KDC or server administrator	used by all MSHN compute resources	KDC defined	• receive job session key
$U_{pub/pri}$	User client key	P	KDC or user	Client Library	KDC defined	• receive job session key • user authentication

Table 1: Key Usage in the MSHN Security Architecture

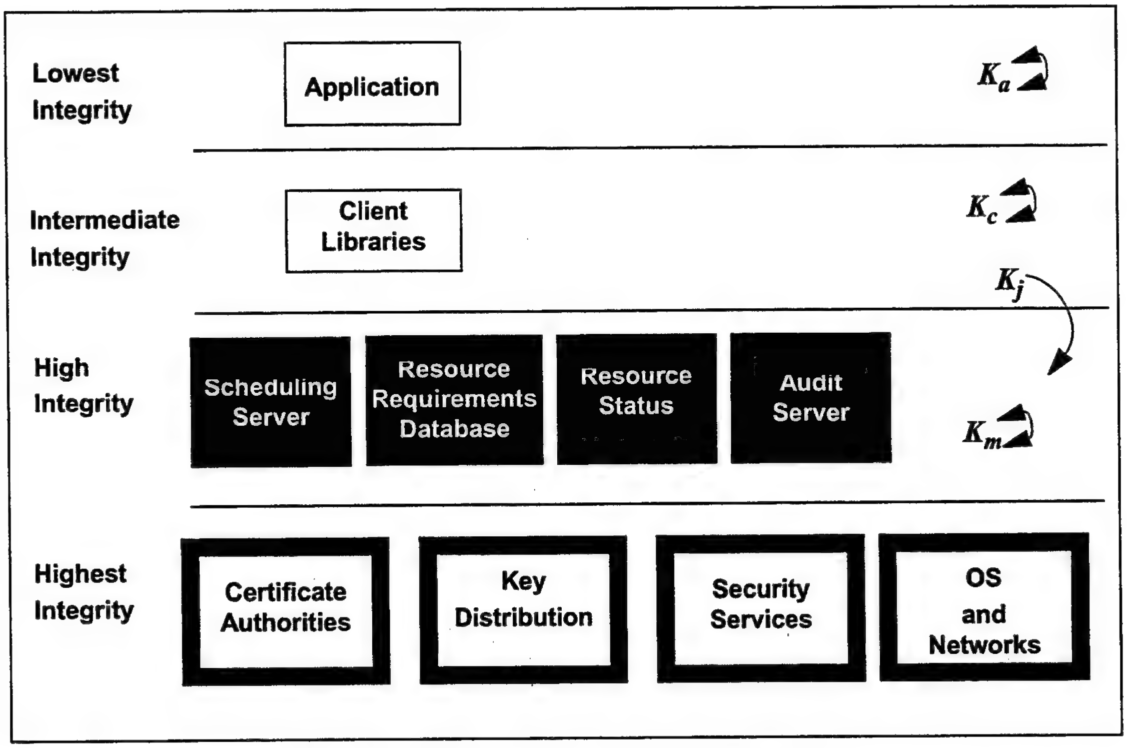


Figure 2. MSHN Security Domain Architecture

the certificate authorities, hence the certificate authorities are the most trusted components of our VHM security architecture. User tasks will be separated on a per-task basis using appropriate keys supplied to each domain. Within MSHN, the highest integrity domain, viz. the MSHN Core, will manage the creation and allocation of session keys used by lower integrity domains. Thus the client domains are issued per-task keys when a job is scheduled and these keys are used for dynamic task management and Core database updates. In addition to being isolated on a per-task basis, each domain is able to distinguish communications from members at its integrity from those of entities of greater or lesser integrity. The MSHN domains are illustrated in Figure 2. Policy is allocated to each of these domains and they are organized hierarchically. This is in keeping with notions of *balanced assurance* for distributed systems [13].

4.1 MSHN Core Domain

The MSHN core domain contains the basic MSHN core services: the Scheduling Server, the Resource Status Server, and the Resource Requirements Database. For the security architecture, we have added an Audit Server. The correct function of these servers is essential to insuring that jobs receive useful scheduling advice; the corruption of either their data or algorithms would render MSHN potentially useless or even detrimental to job performance and quality of service. Thus we assign to these system elements a high integrity level.

The MSHN Scheduling Server will make scheduling decisions based upon the conjunction of authorization information and other scheduling information. When a scheduling request is issued to MSHN, the Scheduling server will first determine the systems accessible by the user. This list of systems will be input to the scheduling algorithms and an advisory schedule will result. Thus for the same job and all other factors being equal, two users could have different schedules depending upon their access to compute resources.

4.2 Client Library Domain

The client libraries will occupy a domain of intermediate integrity. A copy of the client libraries will be available to each user's job. Although users might share the text of client libraries, viz. the client library code, when executing on the same machine, each instance will have separate data. Client libraries will also be able to securely communicate with client libraries on other compute servers. This will permit coordination of parallel processing or checkpointing and movement of an application to a different compute server.

4.3 Application Domain

This domain is intended for user jobs. Each application will be distinct. Distributed, security-aware applications may contain their own end-to-end mechanisms or may use underlying security services.

A legacy application need not become "security aware" in order to be protected. When users request scheduling and process management services of MSHN, they may include protection services as part of the QoS specification for their jobs. The scheduling advisor will determine the best way to meet those protection needs within the VHM accessible by the user. Along with the scheduling recommendations returned to the user, algorithms and keys held by the underlying layers will be available to protect communications for distributed applications. We note that exposure of keys to applications is not intended: either keys are cryptographically protected or completely hidden by the underlying mechanism.

4.4 Mechanisms

Cryptographic mechanisms will be used to isolate and protect the elements of the MSHN architecture in "lightweight" domains. Table 1 describes the major keys used in our design. Key types are for symmetric, S, and public/private, P, cryptography. We intend to support updates of session keys during job runs. The frequency of these updates will be defined as part of the quality of service for security. It is expected that, as the public key infrastructure matures, key distribution centers (KDCs) will issue public/private key pairs. In the interim, individual systems may be required to act as their own KDCs.

4.4.1 Core Keys. The MSHN core will be assigned its own set of keys. Each MSHN core server will have a public/private key pair, $C_{pub/pri}$. These key pairs will be issued by a key distribution center (KDC) and can be verified by a certificate authority (CA) for inter-component authentication.

We believe that symmetric key cryptography provides the most efficient technique for the frequent communications that will be required between MSHN core components. Core components associated with a particular instance of MSHN share a MSHN-wide session key, K_m. This key permits the use of fast symmetric algorithms for the efficient transmission of core-relevant data.

A question that arose in our analysis was how to handle replicas of MSHN core services. This might occur in situations where the system is highly distributed and replicas of one or more MSHN core services are placed at strategic points in the network. Since it is desirable that a client be able to send updates to any one of the replicated servers of a particular type, e.g., Resource Status Server, with a consistent result across the entire system, the session keys of clients could be available to all status servers. Alternatively, dynamic key exchange algorithms could provide protected communications between core components.

4.4.2 Client and Application Keys. The MSHN Core will be responsible for issuing session keys for each job. These keys will be protected enroute to the compute resource using public key algorithms with $R_{pub/pri}$. There will be three session keys associated with each job: a Job Session Key, K_j, a Client Library Session Key, K_c, and an Application Session Key, K_a.

The Job Session Key will be used by the compute client to transmit status information back to the MSHN Resource Status Server and to the Audit Server. It will be used by the MSHN Scheduler to transmit advisory information dynamically to the user client and the compute client. For example, information regarding dynamic job adaptation can be transmitted to the compute client. A database tying user IDs to job IDs and session keys will be distributed across relevant MSHN core servers such as the Scheduling Server and the Audit Server.

The Client Library Session Key will be used for intra-domain management of clients associated with a particular job both at the user's platform and at the compute resources.

The Application Session Key is provided for the benefit of the application. It represents the keys and algorithms proposed by the MSHN core to meet the protection requirements specified by the user for the particular instance of the job run.

4.4.3 User Authentication. Individual users will have public/private keys, $U_{pub/pri}$. User's public keys will be stored in certificates which are available to all elements of the system through a directory service. The MSHN Scheduler will be able to authenticate the user using public key technology. The user will digitally sign messages using his private key and the scheduler will validate the source of the signed message with the user's certified public key.

5. Prototype Demonstration

A prototype demonstration of the MSHN security architecture has been implemented [20]. In keeping with the emphasis on commercial-off-the-shelf PCs embraced by the Navy's IT-21 initiative [7], our demonstration is implemented on three Windows NT Platforms. One platform served as the user interface from which jobs were submitted. A second platform hosted the MSHN Core database and scheduling services, each executing as a separate process. The last platform played the role of a compute resource. (Clearly, in a complete implementation compute resources might be high performance processors.)

For this demonstration we chose all certificates to be homogeneous and selected X.509 v3 certificates [6] for use throughout the VHM for authentication mechanisms.

5.1 Security Layer

One of our principle concerns has been the provision of security services within the context of a well engineered, modular architecture. Toward this end, we have defined a set of security services to be provided to MSHN core components and clients. This thin service layer allows us to achieve the support for heterogeneous platforms sought by the project. Also, it permits us to postpone API choices so that services presented by several APIs can be used, such as CDSA [3] and GSS-API [1]. Thus, a variety of security implementations can be used and these will be transparent to the VHM core, clients, and applications.

The security services layer consists of the following interfaces exported to MSHN core services and client libraries:

mshn_sl_init Initialize databases for MSHN security layer and create any necessary security contexts.

mshn_sl_create_cert Create a certificate with the specified parameters

mshn_sl_get_cert Obtain a certificate for the specified principle

mshn_sl_cert_verify Check that the specified certificate is valid

mshn_sl_cert_revoked Determine if the specified certificate has been revoked

mshn_sl_get_public_key Return the public key for the principle associated with the specified certificate

mshn_sl_get_private_key Return the private key for the specified subject. Note that, in the implementation, the private key is not actually returned, but a handle to the key is provided and this handle is used as a parameter to encryption or decryption functions. Thus the private key is never exposed.

mshn_sl_put_audit Write a record to the audit file or server

mshn_sl_encrypt Encrypt the specified data using the specified key and algorithm

mshn_sl_decrypt Decrypt the specified data using the specified key and algorithm

mshn_sl_sym_key_gen Create a key to be used with symmetric encryption and decryption.

mshn_sl_asym_key_gen Create a public/private key pair to be used with asymmetric encryption and decryption. Only information about the private key is exported, the key itself is not exposed.

mshn_sl_msg_digest Hash the specified buffers and generate a message digest.

mshn_sl_sign Produce a signature for the specified data in two steps: create a message digest and then encrypt the message digest.

mshn_sl_sig_verify Verify a digital signature for a specified data-signature pair.

5.2 Common Data Security Architecture

The Intel Common Data Security Architecture (CDSA) has been used as the basis for a proof of concept demonstration of the MSHN core services.

CDSA is intended to provide a basic security infrastructure for use with personal computers. It is a layered architecture and is intended to be modular, portable and adaptable. It has currently been implemented on Windows NT, a system of particular interest to the U.S. Navy as a result of its IT-21 initiative. An implementation of CDSA by RSA Security is underway and will provide the components on Unix platforms. In the interim, we have implemented a subset of the MSHN Security functions on a Unix platform using SSLeay encryption software[22] (a public implementation of SSL [9]).

CDSA consists of three primary layers:

- a set of system security services
- the Common Security Services Manager (CSSM)
- add-in security modules.

The add-in security modules are organized by the CSSM so that service provider interfaces (SPIs) can be defined for each module job manager.

Services are separated into four categories:

cryptographic services: An add-in cryptographic service module can provide the following functions

- bulk encryption and decryption
- creation and verification of digital signatures

- cryptographic hash creation
- key generation
- random number generation
- encrypted storage of private keys. We note here that ultimately the protection of keys, must depend upon a system-based protection mechanism.

trust policy services: The intent of the trust policy services is to provide access control over various system activities. Certificates are presented to the trust policy manager and access decisions are made according to information contained in the certificate.

certificate services: For a PC, the certificate services provide a self-contained certificate management mechanism. This module provides support to create new certificates, create certificate revocation lists (CRLs), sign and/or verify certificates and CRLs, import and export certificates created using other formats, extract information such as public keys from certificates, reinstate revoked certificates and search CRLs. The flexibility of these services with respect to certificate format makes them particularly attractive during the current period of evolving standards. For our current work, we have chosen the X.509 v3 format.

data store services: These modules are intended to provide secure and persistent storage for non-volatile information such as certificates and CRLs. Clearly in a system where assurance is of significant importance, data store services would be relegated to an underlying high assurance TCB and the access control mechanisms of the TCB would be used to ensure that only authorized users (or processes acting on behalf of those users) had modify access to security-critical databases.

6. Other Work and Future Plans

The Sigma project is addressing heterogeneity on the enclave level with its exploration of Internet Interoperability Protocol gateways to support Common Object Request Broker Architecture (CORBA) [2] interoperability [17]. This work is not directly integrated into a VHM management system and provides only a highly coarse level of access control and enclave protection. However, this effort could be viewed as complementary to ours as the MSHN security mechanisms could be used in an environment supporting Sigma controls.

Globus [8] is using a proxy-based system to permit computation on behalf of users on remote resources. The mechanism bears a remarkable similarity to Kerberos [18] and currently suffers from several scalability and trust domain drawbacks. Our approach provides a more straight forward path toward accountability.

Legion [21] claims to permit users to define policies at the granularity of objects. It is *ad hoc* with respect to privilege in that privilege is minimized everywhere -- no element of the system is responsible for any element other than itself. In addition, Legion does not provide for certificate authorities or key distribution centers. Instead, Legion-specific unique object identifiers are required globally to support key management. It appears that instead of supporting interoperability with respect to key management, Legion is dictating a homogeneous approach.

6.1 Future Work

Quality of service and security can sometimes be regarded as conflicting rather than complementary goals [10]. During crises, it is likely that demands upon a system may be significantly higher than during normal operations. This may mean that tasks may be required to adapt in order to provide a minimal level of service. We plan to explore the possibility of providing information to users and administrators regarding the performance costs incurred by using security measures commensurate with the sensitivity and reliability associated with the task. Hence the security requirements of different tasks can be met with measures of differing "strengths." At this juncture, we are not suggesting that security measures can be relaxed for a job with an assigned sensitivity and reliability.

Another area of future activity will be in the development of an audit mechanism for the VHM. As noted earlier, we plan to use a combination of audit and digital signatures as the cornerstone of non-repudiation services in our VHM.

Lacking a Unix implementation of CDSA, a Unix version of our prototype was created using SSLeay. In a fully layered architecture, SSL might depend upon CDSA for security support.

7. Summary

This paper has described a security architecture intended to support a virtual heterogeneous machine. A consistent set of choices must be made to construct data structures and mechanisms to use the underlying environment effectively. We build assurance into the system by relying on underlying mechanisms rather than constructing our own.

Our architecture for the MSHN VHM consists of four domains: the underlying operating sytsem, MSHN Core Services, Client Services, and Application. Domains start with their own identification and authentication evidence provided by key distribution centers and certificate authorities. When the VHM is started, the Core Services establish their own cryptologically defined domain. Client Services submit job requests to the Core and are provided with session keys which establish per-session domains that permit communication between Clients and Core Services. Using underlying services, Clients create domains on behalf of Applications, to which they provide handles to keys for application-level communications security.

Future work will integrate quality of service mechanisms into this architecture and explore the creation of high assurance audit mechanisms.

Acknowledgements

We wish like to thank Timothy Levin for his critical reading of the manuscript and helpful comments.

References

[1] Global Security Services Application Programming Interface, RFC 2078.

[2] The Common Object Request Broker: Architecture Specification, December 1993.

[3] Common Data Security Architecture Specification, Intel Corporation, Santa Clara, CA release 1.2 edition February 1998. http://developer.intel.com/ial/security/.

[4] Anderson, J.P., Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, Airforce Electronic Systems Division, Hanscom AFB, Bedford, MA, 1972. (Also available as Vol. I, DITCAD-758206, Vol. II DITCAD-772806).

[5] Bricker, A., Litzkow, M.,l and Livny, M., Condor Technical Summary, Technical Report CS-TR-92-1069, Computer Science Department, University of Wisconsin, Madison, WI, January 1992.

[6] CCITT. Recommendation X.509: The Directory - Authentication Framework, 1989. also ISO/IEC 9594-8.

[7] Clemmins, A., IT-21: The path to Information Superiority, CHIPS, July 1997.

[8] Foster, I., Karonis, N., Kellelman, C., Koenig, G., and Tuecke, S., A Secure Communications Infrastructure for High-Performance Distributed Computing, In *Proceedings 6th IEEE Symposium on Distributed Computing*, pages 125-136. IEEE Computer Society Press, 1997.

[9] Freier, A., Karlton, P., and Kocher, P., *The SSL Protocol, Version 3.0*, Netscape Communications, March 1996. URL: http://home.netscape.com/eng/ssl3/index.html6.

[10]Greenberg, I., Lunt, T., Clark, R., and Wells D., Secure Alpha: Security Policy and Policy Interpretation for a Class B3 Multilevel Secure Real-Time Distributed Operating System, Technical Report ELIN A003, SRI International, Menlo Park, CA, April 1993.

[11]Hensgen, D., and Kidd, T., MSHN design documents. note, February 1998.

[12]Irvine, C. E., Acheson, T.B., Thompson, M. F., Building Trust into a Multilevel File System, In *Proceedings 13th Nation Computer Security Conference*, pages 450-459, Washington, DC, October 1990.

[13]Irvine, C.E., Schell, R. R., and Thompson, M.F., Using TNI Concepts for the Near Term Use of High Assurance Database Management Systems. In *Proceedings Fourth RADC Database Security Workshop*, Little Compton, RI, April 1991.

[14]Kidd, T., Hensgen, D., Freund, R., Moore, L., SmartNet: A Scheduling Framework for Heterogeneous Computing, In *International Symposium On Parallel Architectures, Algorithms, and Networks*, Beijing, China, June 1996.

[15]Kresho, J.P., Quality Network Load Information Improves Performance of Adaptive Applications, M.S. thesis, Naval Postgraduate School, Monterey, CA September 1997.

[16]Litzkow, M., Tannenbaum, T., Basney, J. and Livny, M., Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System, Technical Report CS-TR-97-1346, Computer Science Department, University of Wisconsin, Madison, WI, April 1997.

[17]Sebes, J. and Benzel, T. C. V, SIGMA: Security for Distributed Object Interoperability Between Trusted and Untrusted Systems, Technical draft paper, Trusted Information Systems, Mountain View, CA 1996.

[18]Steiner, J., Neumann, B. and Schiller, J., Kerberos: An authentication System for Open Network Systems, Usenix Conference Proceedings, pages 191-202, 1998.

[19]Stevens, R., Woodward, P., DeFanti, T., and Catlett, C., From the I-Way to the National Technology Grid, Comm. A.C.M., 40(11):51-60, 1997.

[20]Wright, R., Integrity Architecture and Security Services Demonstration for Management System for Heterogeneous Networks, M.S. Thesis, Naval Postgraduate School., Monterey, CA, June 1998.

[21]Wulf, W., Wang, C., and Kienzle, D., A New Model of Security for Distributed Systems, UVa CS Technical Report CS-95-34, University of Virginia, Richmond, VA, August 1995.

[22]Young, E. A., SSLeay, June 1997. ftp://ftp.psy.uq.oz.au/ pub/Crypto/SSL.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library, Code 013 2
Naval Postgraduate School
Monterey, CA 93943-5100

3. Research Office, Code 09 1
Naval Postgraduate School
Monterey, CA 93943-5138

5. Dr. Cynthia E. Irvine 10
Code CS/Ic
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5118